Fanatical Support for AWS Product Guide

Rackspace

June 26, 2020

CONTENTS

Last updated: April 30, 2020

IMPORTANT: This is a PDF version of the Product Guide, and is intended to be used for point-in-time offline reference purposes only. The authoritative version of this document lives online at https://developer.rackspace.com/docs/fanatical-support-aws and will contain the latest updates.

This Product Guide is designed to provide a detailed look at how Rackspace delivers our **Fanatical Support for AWS** offering. It covers core concepts such as the *AWS account structure* and Rackspace *service levels*, and advanced concepts such as provisioning bastion access via Rackspace Passport and accessing audit logs via *Rackspace Logbook*.

For general information on the offering, please visit https://www.rackspace.com/aws.

To sign up, visit https://cart.rackspace.com/aws.

CHAPTER

ONE

GETTING STARTED

It is extremely easy to get started experiencing Fanatical Support for AWS.

1.1 Create your Rackspace account

The first step is to create your Rackspace account. Visit https://cart.rackspace.com/aws and follow the instructions to establish your account.

1.2 Add a new AWS account

Once you have created your Rackspace account navigate to the Fanatical Support for AWS Control Panel. Login using the credentials you established during the signup process above.

Note that all new Rackspace accounts undergo a thorough review to minimize fraud. This process can take several minutes to several hours to complete, depending on the details of your signup. You will not be able to proceed until the verification is complete. If you would like to expedite the verification process, please *Contact Us*.

Once you are logged in you will see an option to add a new AWS account. Provide the relevant details and select a *service level*. We will immediately provision you with a new AWS account ready for your use. You can click the "Log in to AWS Console" link to go straight to AWS, though we encourage you to first review our *Recommended Network Configuration*.

If at any time you need assistance from a Racker, please do not hesitate to *Contact Us*.

1.3 Use an existing AWS account

Our general recommendation is to create a new AWS account as it will be provisioned immediately and will already include all of our best practice configuration ready for you to use. If you have an existing AWS account that you would like to use with our services, please see *Transferring existing AWS accounts to Rackspace* for additional details regarding the process.

CHAPTER

TWO

AWS ACCOUNTS

Each Rackspace account can house one or more AWS accounts. By default, you can create up to five new AWS accounts via the Fanatical Support for AWS Control Panel. If you need more than five accounts, please open a ticket to request a limit increase. In addition to creating new AWS accounts, you may also *transfer existing AWS accounts* to Rackspace for management.

Each AWS account provides a top-level administrative control boundary for the resources that are a part of it. While it is possible to leverage Amazon's Identity and Access Management (IAM) platform to isolate certain resource access, we typically recommend provisioning an AWS account per application deployment phase (e.g. development, staging, and production), thereby allowing you to assign different users in your organization access to one or more of the accounts without complex IAM policies. In this example, developers could be granted access to provision EC2 instances, RDS databases, etc. in your development and staging accounts, but be restricted to read access of the resources in your production account.

In addition to being a strong permission boundary, AWS accounts also provide a convenient construct for tracking expenses, since by default, both AWS and Rackspace charges are grouped by AWS account. For example, if 4 separate AWS accounts are used called app1-dev, app1-prod, app2-dev, app2-prod, it is very easy to see how much is being spent on each application environment. We highly encourage the use of tagging for more fine grained tracking of expenses within accounts, but tagging is more complicated, certain resources may be missing tags resulting in unallocated cost, and not all AWS resource types support tagging. AWS accounts provide a great default cost allocation construct.

Lastly, using separate AWS accounts per environment gives you the flexibility to select different Rackspace *service levels* for each environment, since Rackspace service levels are applied at the AWS account level. For example, you may opt for the Navigator service level on your development account while using the Aviator service level for your production environment.

As is described later in this document, several Fanatical Support for AWS features (such as *Rackspace Logbook*) are available in both cross-account and account-specific views, enabling unified visibility across multiple AWS accounts.

2.1 Account Defaults

For all AWS accounts managed by Rackspace, whether created new via the Fanatical Support for AWS Control Panel or created directly with AWS and transferred to Rackspace, we automatically apply several default settings to the account based on best practices we have developed in cooperation with AWS. You should not change or disable any of these default settings, as they are critical to our delivery of Fanatical Support.

- AWS IAM (Identity and Access Management)
  - Setup an IAM role named "Rackspace" for ongoing access to the account (see *AWS Identity and Access Management (IAM)* for additional details)
  - Set the IAM account password policy for all passwords
    - At least 12 characters in length

        * Contain at least one uppercase character
        * Contain at least one lowercase character
        * Contain at least one number
        * Contain at least one symbol
        * Not one of the previous 24 passwords used
    - Create an IAM role named "AWSConfig" for use by the AWS Config service
    - Create an IAM role named "RackspaceTools" to allow us to provide you with Compass
    - Create an IAM role named "RackspaceDefaultEC2Role" along with an attached IAM policy named "RackspaceDefaultEC2Policy" which can be attached to EC2 instances to provide access to AWS Systems Manager and the CloudWatch EC2 Agent.
    - Create an IAM role named "rackspace-passport-bastion" with an IAM policy that allows bastion logs to be written to CloudWatch Logs.
    - Create an IAM role named "EC2ActionsAccess" for use by CloudWatch alarms to trigger actions on instances.
    - Create an EC2 Instance Profile named "rackspace-passport-bastion-instance-profile" with the "rackspace-passport-bastion" role attached.
- AWS S3 (Simple Storage Service)
    - Create a bucket named "<account_uuid>-logs" in the US West 2 (Oregon) region
        * Enable versioning and apply an S3 bucket lifecycle policy to the "<account_uuid>-logs" bucket that expires files after 365 days and permanently removes deleted files after 90 days
        * Set an S3 bucket policy on the "<account_uuid>-logs" bucket to allow write access from CloudTrail
    - Create a bucket named "<account_uuid>-ssmoutput" in the US West 2 (Oregon) region
        * Apply an S3 bucket lifecycle policy to the "<account_uuid>-ssmoutput" bucket that deletes files after 60 days
- AWS CloudTrail
    - Configure AWS CloudTrail in each AWS region to log to the S3 bucket named "<account_uuid>-logs"
    - Configure an SNS topic named "rackspace-cloudtrail" in each region and subscribe it to a region-specific Shared Management Services SQS queue for use by the *Rackspace Logbook* service
- AWS CloudWatch
    - Create a "rackspace-passport-bastion" CloudWatch Logs Group where logs are shipped for Rackspace operational purposes.
- AWS Config
    - Configure AWS Config in each AWS region to log to the S3 bucket named "<account_uuid>-logs"
    - Configure an SNS topic named "rackspace-awsconfig" in each region and subscribe it to a region-specific Shared Management Services SQS queue for use by Rackspace tooling
- AWS SNS (Simple Notification Service)
    - Create SNS topics named "rackspace-support", "rackspace-support-standard", "rackspace-support-urgent", "rackspace-support-emergency" in each region and subscribe it to a region-specific Shared Management Services SQS queue for use by our *Rackspace Watchman* service

2.2 Transferring existing AWS accounts to Rackspace

While the Fanatical Support for AWS Control Panel enables the ability to easily provision new AWS accounts, there may be situations where you would like to transfer an existing AWS account to Rackspace for management. This is also supported, and once complete, will allow Rackspace management tooling and expertise to function with your existing AWS account.

This process involves formally assigning your AWS account to Rackspace for management, which can be initiated by submitting a request via the Fanatical Support for AWS Control Panel. Click the **Add AWS Account** card at the bottom of the AWS account list. For account source, select **Use an existing AWS account not currently managed by Rackspace**. The following information is required:

- AWS Account Number
- Legal Company Name
- Legal Company Address
- Authorized Signatory Name (the individual who can legally give authorization to assign your AWS account to Rackspace)
- Authorized Signatory Email Address

Once we receive your request, we will create a ticket for you with instructions that your team must carry out in order to prepare the AWS account for transition to Rackspace support.

Once your team completes those instructions from the ticket, Rackspace sends a request to AWS to review and approve the account assumption request. AWS will confirm if any custom legal or pricing terms exist that need to be transferred to Rackspace. Once completed, AWS will approve the account assumption.

After those steps are complete, Rackspace will automatically apply several default settings to the account based on best practices we have developed in cooperation with AWS. For details, please refer to the *Account Defaults* section of this product guide.

This process typically takes 2-4 weeks from start to finish, which is somewhat dependent on you since certain steps of the process require action on your part. Please monitor your email and the Support Tickets section of the Fanatical Support for AWS Control Panel for tickets that require your action.

Note that transferring an existing AWS account to Rackspace does not count against the limit of new AWS accounts you are able to provision via the Fanatical Support for AWS Control Panel.

Please note that Reserved Instance and Savings Plan sharing between AWS accounts may be disrupted during the account transition process. For details, please consult your Rackspace Onboarding Manager or Customer Success Manager.

2.2.1 Minimum Account Requirements

In order for an existing AWS account to be transitioned to Rackspace, it must meet our minimum account requirements, which include:

- No access keys exist for the root user of the AWS account
- The account is not consolidated under a payer account or serving as a payer account with linked child accounts

These requirements **must** be met before the account can be transitioned to Rackspace.

2.3 Offboarding

While we hope to serve you for life, should you ever decide that you no longer require Rackspace's management of your AWS account we can work with you to transition your account to a direct relationship with AWS. You would retain access to all AWS resources, but would lose access to Rackspace tooling such as *Logbook* and Compass as well as Rackspace's AWS expertise and service. If you are considering making this change, please *contact your Account Manager* for further assistance.

During the offboarding process, all Rackspace SNS Topics and IAM Roles will be removed, with the exception of "RackspaceDefaultEC2Role" IAM Role. This role might be used by your EC2 instances. You can keep using this role or you can remove it if it's not used by any instances.

2.4 AWS root credentials and account access

Rackspace requires access to each AWS account's root credentials to complete certain elements of the account transition and maintenance. Root credentials are stored in a secure vault, and access is limited to a specific set of employees. This access is audited and constrained to only the few situations in which the credentials are required. Rackspace will transfer control of these credentials to your ownership should the partnership with Rackspace end.

Why must Rackspace hold root credentials for AWS accounts?

Securing AWS accounts to prevent fraudulent usage, and security compromise is a critical aspect to our management of AWS accounts. AWS specifies a number of Identity and Access Management best-practices to minimize the risk of account compromise. These include locking away the AWS account root user access keys and enabling multi-factor authentication (MFA). By holding the root permission/credentials, we are enabled to enforce these best practices on AWS accounts.

Amazon requires Rackspace to use these root credentials to communicate with Amazon's billing department to address billing issues.

The root user's email address is set to a unique Rackspace email account. Automation creates and updates tickets in the Rackspace ticketing systems based on emails sent to that email address. This lets Rackspace Fanatical Support for AWS (FAWS) to share important account updates with you via ticket.

How will my team and I be impacted by not holding our AWS account root user credentials?

You'll be able to manage all access as either:

- Users within the Fanatical Support for AWS Control Panel
- IAM Roles for AWS resources, such as EC2 instances (see AWS Identity and Access Management)

For the specific tasks that require AWS account root user permission, a small set of the Fanatical Support for AWS Rackers carry out the tasks.

How does Rackspace secure and store root credentials?

After credentials for the root user of the AWS account are updated to a Rackspace email address, a new strong password is generated, two factor authentication is enabled for the AWS account. The new password is encrypted and vaulted using AWS Key Management Service (KMS).

Nearly all management of AWS accounts by Rackspace tools and employees is accomplished using IAM roles and users.

Why does Rackspace require root access keys to be deleted?

Access keys (an access key ID and secret access key) can be used to make programmatic requests to AWS. Rackspace requires that root access keys be deleted, as using root access keys violates AWS Identity and Access Management best-practices, and exposes the account to risk of security compromise. Amazon encourages the use of IAM users, groups, and roles to manage access.

How does Rackspace release root credentials?

There are two relevant scenarios: the closure of an account, and a reverse assumption (customers taking ownership of accounts). In the case of closing accounts, root credentials will lead to a “dead end” and do not need to be destroyed.

When customers terminate management services with Rackspace, a process is followed that Rackspace refers to as “reverse assumption” process. This includes changing the root user credentials to those defined by the customer.

CHAPTER

THREE

SERVICE LEVELS

Fanatical Support for AWS combines tooling and automation with human experts to deliver a world-class experience. We offer two service levels, Navigator and Aviator, which are selected for each AWS account we support.

- Navigator: "I want to do most things myself, but I want access to Rackspace's AWS experts and tools."
- Aviator: "I want Rackspace to operate and manage my AWS environments for me or with me."

Navigator and Aviator are now legacy service levels that existing customers can maintain on their accounts.

Starting in July 2019, Service Blocks are the standard support offers. Please see the *Service Blocks* section of the Product Guide for further information about these offers.

For details on what is included in each service level, including details on levels of support for each AWS service, download our Fanatical Support for AWS Service Overview - Navigator and Aviator Version or the Fanatical Support for AWS Service Overview - Service Blocks Version.

3.1 Features: Tooling and Automation

A curated set of Rackspace developed and best of breed AWS ecosystem tools:

- AWS Account Generation Pre-Configured with *Rackspace Best Practices*
  - Service Levels: Navigator and Aviator
  - Features
    - AWS root account credentials encrypted and locked away
    - MFA enabled on root account and secret configuration key encrypted and locked away
    - No named IAM users; all AWS access via single, dynamically scoped IAM role and temporary STS credentials
    - CloudTrail and AWS Config enabled with centralized logging
    - Separate AWS accounts per environment (e.g. development, staging, production)
- Access to AWS Trusted Advisor
  - Service Levels: Navigator, Aviator, Platform Essentials
  - Features
    - Access to all Trusted Advisor checks
- *CloudHealth*
  - Service Levels: Navigator, Aviator, Platform Essentials

  - Features
    - Cost and usage visualizations
    - Savings recommendations
    - Governance through policies and actions
- Rackspace Passport
  - Service Levels: Aviator, Manage & Operate (customer and Rackspace use)
  - Features
    - On-demand provisioning of bastions for secure network access to VPC resources
    - Automatic, temporary credential management via the In-Instance Credential Management Service
    - Full logging
- In-Instance Credential Management Service
  - Service Levels: Aviator, Manage & Operate (customer and Rackspace use)
  - Features
    - Automatic certificate authority and SSH key rotation across your fleet of EC2 instances
    - Temporary, fast expiring keys with silent renewal
- *AWS Instance Scheduler*
  - Service Levels: Aviator, Manage & Operate
  - Features
    - Deployment, configuration and management of AWS Instance Scheduler
    - Allows configuration of custom start and stop schedules for EC2 and RDS instances
    - Provides cost saving on environments that aren't used 24/7

3.2 Features: Human Experts

Tap into an army of certified AWS architects and engineers ready to deliver Fanatical Support to your business 24x7x365. Available via ticket and phone.

- AWS best practice and architecture consultation from 100% AWS certified experts
  - Service Levels: Navigator, Platform Essentials (standard use cases) and Aviator, Manage & Operate (customized to your specific application)
- Hands-on management and assistance for all supported AWS services
  - Service Levels: Aviator, Manage & Operate
- EC2 operating system management
  - Service Levels: Aviator, Manage & Operate
  - Features
    - Amazon Linux 2 & Amazon Linux (legacy), Red Hat Enterprise Linux: 6, 7 & 8, CentOS: 6, 7 & 8, Ubuntu LTS Versions: 16.04 & 18.04, Windows Server 2008 R2*, Windows Server 2012 R2, Windows Server 2016, Windows Server 2019.
    - Configuration, Optimization, Patching, Upgrades

  - Prerequisites: The following agents must be installed and working on your

EC2 instances in order to be supported by Rackspace

  - Passport - The server agent allows Rackspace support team to access your instances via SSH (Linux) or RDP (Windows)
  - SSM - The AWS Systems Manager agent allows Rackspace to manage your EC2 instances remotely (instance configuration, maintenance of agent versions and updates, OS patching, software inventory monitoring)

- *Rackspace Watchman*
  - Service Levels: Aviator, Manage & Operate
  - Features
    - Rackspace AWS certified engineer response to CloudWatch alarms 24x7x365
    - Set up CloudWatch alarms to a pre-configured SNS topic or let us do it for you
- Custom CloudFormation template creation
  - Service Levels: Aviator, Architect & Deploy
- Data restoration support (for EC2 and RDS exclusively)
  - Service Levels: Aviator, Manage & Operate

* Support for Windows Server 2008 R2 is contingent on enabling an alternative means of access for Rackspace to manage your instances. Please work with your Support team prior to deploying new instances running Windows Server 2008 R2.

3.3 Response Time SLAs

Rackspace will respond to your support requests submitted to us via ticket in the following timeframes. All requests should be made directly to Rackspace and we will escalate to AWS directly, if needed.

- Emergency (Business-Critical System Outage / Extreme Business Impact): If Rackspace Infrastructure monitoring and alerting services determines your AWS Services are inaccessible from the public internet, which may result in the inability to complete business transactions, our initial response to emergency monitoring alarms will occur within fifteen minutes (Aviator service level only; monitoring response is not included in the Navigator service level).
- Urgent (Production System Outage / Significant Business Impact): If your AWS Services are functioning improperly or at less than optimal performance and the failure is impacting business transactions, our initial response is 60 minutes. Customers must call Rackspace immediately after creating the Urgent ticket to trigger the one hour response guarantee. This severity is only available for the Aviator service level.
- High (Production System Impaired / Moderate Business Impact): If your AWS Services are functioning improperly or at less than optimal performance, but the failure is not impacting business transactions, our initial response to your support request submitted to us via a ticket will occur within four hours at the Aviator or Navigator service levels.
- Normal (Issues and Requests / Minimal Business Impact): If your AWS Services are functioning normally but you have a time sensitive request, question, or issue that needs addressed, our initial response to your support request submitted to us via a ticket will occur within 12 hours at the Aviator and Navigator service levels.

- Low (General Information, Questions, and Guidance): If your AWS Services are functioning normally but you require information or assistance, wish to schedule maintenance, or require the completion of any other non-immediate tasks, our initial response to your support request submitted to us via a ticket will occur within 24 hours at the Aviator and Navigator service levels.

3.4 Supported Operating Systems

Fanatical Support for AWS supports the following operating systems (OSs):

Operating System	Supported Until
Amazon Linux 2	June 30, 2023
Red Hat® Enterprise Linux® (RHEL) and CentOS® 7	June 30, 2024
RHEL and CentOS 8	May 2029
Ubuntu 16.04 LTS	April 30, 2021
Ubuntu 18.04 LTS	April 30, 2023
Ubuntu 20.04 LTS	April 30, 2025

The following OSs are approaching End of Life (EOL) as determined by the OS vendor:

Operating System	EOL Date
RHEL and CentOS 6	November 2020
Amazon Linux (legacy)	December 2020
Ubuntu 16.04 LTS	April 2021

CHAPTER

FOUR

PRICING

Your monthly service fees will be calculated by pooling the AWS infrastructure charges from all of your AWS accounts at the same service level. Learn more about monthly service fee calculations in the *Billing* section.

Service fees are charged in addition to AWS infrastructure rates. Service Fees for AWS Reserved Instances and Spot Instances are calculated based on the corresponding on-demand list rates. AWS infrastructure is charged at the list rates on the AWS website. You can view your service fees in the Fanatical Support for AWS Control Panel.

Please note that the AWS free tier is not available to Fanatical Support for AWS accounts.

CHAPTER

FIVE

AVIATOR INFRASTRUCTURE MANAGEMENT

5.1 Summary

Based on your desired infrastructure management preference, some AWS environments built under the Fanatical Support for AWS Aviator service level are managed directly via the AWS console, while others are managed through a process known as Infrastructure as Code (IaC), specifically using an AWS service named CloudFormation.

5.2 Management via the AWS console

Organizations that are not used to Infrastructure as Code (IaC) practices can benefit from a simplified management of their AWS environment via the AWS console. With this type of management, small changes can be implemented quicker. If your environment is managed via the AWS console, you have the added flexibility to make changes by yourselves, if you wish to do so. However, new resources need to be deployed by Rackspace.

5.3 Management via IaC using CloudFormation

Organizations that are comfortable with IaC practices can have their AWS environments managed using CloudFormation. With this method of management, all changes must be requested via tickets and deployed by Rackspace. Changes via the AWS console are not allowed since they can conflict with CloudFormation management, resulting in downtime, data loss, or delays to reconcile these manual changes. It is important that all changes to your environment are managed with CloudFormation.

5.4 Why use CloudFormation?

If you are comfortable working with a strict change management process and giving up the ability to make changes via the AWS console, managing your environment using CloudFormation can provide the following benefits:

- Ability to automatically and consistently rebuild or duplicate environments
- Version-controlled and quality checked infrastructure changes
- Automated testing of the interconnected parts of an environment
- Inherent Disaster Recovery plans for your infrastructure

5.5 What resources are managed with CloudFormation?

If your environment is managed via IaC, you should consider all of it under the control of CloudFormation, unless the Rackspace Support team instructs you otherwise.

The following resources are never maintained with CloudFormation, however we still recommend engaging the Rackspace Support team for any changes!

- IAM User Console Passwords
- IAM User Access Keys
- Elastic Beanstalk Environments
- Route 53 External Zones

Resources in the list above can be modified using the AWS console without the risk of creating configuration drift. If you're unsure, please contact the Rackspace Support team.

5.6 What if I make changes outside of CloudFormation?

If your environment is managed via IaC, making changes outside of CloudFormation creates configuration drift, meaning that the code we use to maintain your environment is no longer in sync with what actually exists within your AWS account. This may result in:

- Downtime or data loss, as manual infrastructure changes are overwritten (configuration changed, resources recreated or deleted) by a CloudFormation stack update
- Delays implementing changes you request, as the CloudFormation templates and parameters must be reconciled with manual infrastructure changes before proceeding

The impact to the infrastructure may be wider than just the directly modified resources, and in some circumstances may require downtime or a rebuild of much of the environment. Even the smallest change can have wide reaching consequences!

If you have been forced to make manual infrastructure changes in an emergency, please contact the Rackspace Support team as soon as possible to document the changes that were applied.

5.7 I don't want to use CloudFormation, can I opt out?

Yes, Rackspace allows your environment to be managed via the AWS console. If you wish to discuss the strategy for ongoing management of your environments, please don't hesitate to contact your Account Manager, who will be happy to discuss the options with you!

5.8 Terraform and GitHub Support (Limited Availability)

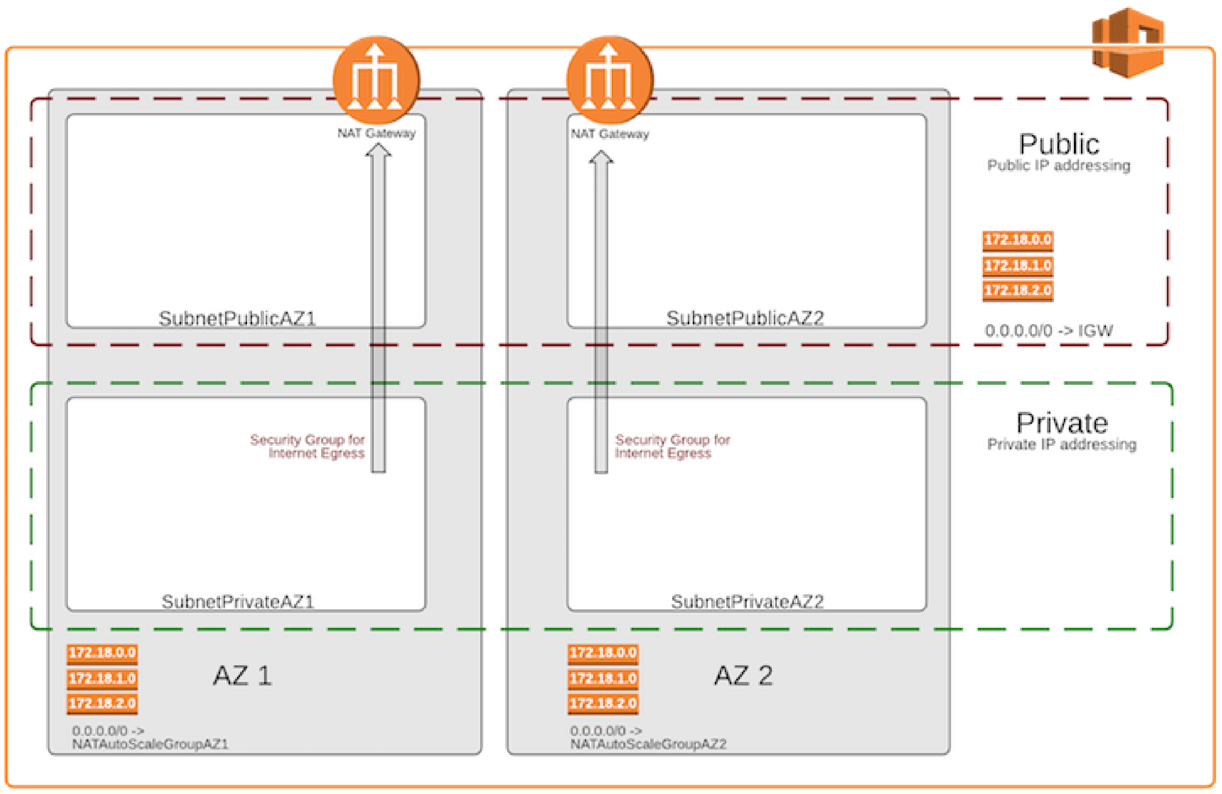
Fanatical Support for AWS offers, in limited availability, support for :ref:'Terraform <using_terraform>' and :ref:'GitHub <using_github>' as an alternative to having your infrastructure managed by CloudFormation. If you're interested in learning more about this future option, please reach out to your Account Manager.

CHAPTER

SIX

RECOMMENDED NETWORK CONFIGURATION

This section describes the necessary scaffolding and processes to create the initial AWS network environment for Rackspace customers using AWS through the Fanatical Support for AWS offering. A CloudFormation template and additional supporting scripts will be used to create the initial network and all of its necessary components, thus providing Public and Private subnets for EC2 instances and other AWS services.



This includes:

- A Single VPC
- Availability Zones (AZ) Options
  - Two AZ deployments are the standard
  - Three AZ deployment to address specific application requirements
- Subnets

  - Public Tier - could be accessible from the Internet
  - Private Tier - could access the Internet via a NAT environment
  - Subnets in each Tier will have the same network masks
- Highly Available Outbound NAT (HA-NAT) with Elastic IP - for EC2 gateways in the Private Subnets
- Security Groups - primary method to isolate and secure workloads
- Tagging - to address Rackspace billing and operational processes

You can access the template by downloading it from here.

Note: The template will create AWS resources for which you will be charged (for example, EC2 NAT gateways).

6.1 CloudFormation

There are two important concepts to understand when using AWS CloudFormation: *templates* and *stacks*. A template is used to describe your AWS resources and their properties. When you create a stack, AWS CloudFormation provisions the resources that are described in the template.

To learn more, view the AWS documentation on stacks and templates.

6.1.1 Rackspace CloudFormation Template: BaseNetwork

In our Aviator *service level* we assist customers with creating custom CloudFormation templates to describe their environments. For customers at both the Navigator and Aviator service levels we make a standardized CloudFormation Template, BaseNetwork, available to create the initial network and all of its necessary components. The rest of this section will describe the elements that are part of the BaseNetwork CloudFormation Template, and their associated components. You can download the BaseNetwork template here.

Parameters

- VPCCIDR - CIDR for the VPC
- SubnetPublicAZ1 - CIDR for Public subnet
- SubnetPublicAZ2 - CIDR for Public subnet
- SubnetPrivateAZ1 - CIDR for Private subnet
- SubnetPrivateAZ2 - CIDR for Private subnet
- InstanceTenancy - Single or Multi-Tenant Hypervisor
- Environment - Dev, Test, Prod etc.

Networking

- The CloudFormation template has two major options:
  - 2 Availability Zones with 4 Subnets
  - 3 Availability Zones with 6 Subnets
- Defaults to using CIDR: 172.18.0.0/16
  - Public Ranges

    * 172.18.0.0/22 - 1,022 Hosts - Public AZ1
    * 172.18.4.0/22 - 1,022 Hosts - Public AZ2
    * 172.18.8.0/22 - 1,022 Hosts - Public AZ3
    * 172.18.12.0/22 - 1,022 Hosts - Public AZx
    * 172.18.16.0/20 - 4,094 Hosts - Additional public (4 more public with size listed above)
  - Private Ranges
    * 172.18.32.0/21 - 2,046 Hosts - Private AZ1
    * 172.18.40.0/21 - 2,046 Hosts - Private AZ2
    * 172.18.48.0/21 - 2,046 Hosts - Private AZ3
    * 172.18.56.0/21 - 2,046 Hosts - Private AZx
    * 172.18.64.0/18 - 16,382 Hosts - - Additional private (8 more private using size above)
  - 172.18.128.0/17 - 32,766 Hosts - Special needs (16 more private using size above)

- Route Tables
  - RouteTablePublic - route table for Public subnets
  - RouteTablePrivateAZ1 - route table for Subnet Private AZ1
  - RouteTablePrivateAZ2 - route table for Subnet Private AZ2
- Default Gateways
  - Internet Gateway (IGW) - Default GW for the Public Subnets
  - ASGNatAZ1 Instance ID - Default GW for Subnet Private AZ1
  - ASGNatAZ2 Instance ID - Default GW for Subnet Private AZ2
  - ASGNatAZ3 Instance ID - Default GW for Subnet Private AZ3 (if necessary)

HA NAT

- High Availability NAT gateways get created in the public subnets (1 per AZ)
  - NatAZ1
  - NatAZ2
  - NatAZ3 (if necessary)

Tags

- Service Provider - “Rackspace”
- Environment - from Parameter Environment
- Name - Resource name (e.g. IGWBase, SubnetPublicAZ2)

Outputs

- outputVPCID
- outputSubnetPublicAZ1
- outputSubnetPublicAZ2
- outputSubnetPublicAZ3 (if necessary)
- outputSubnetPrivateAZ1

- outputSubnetPrivateAZ2
- outputSubnetPrivateAZ3 (if necessary)

6.2 Virtual Private Cloud (VPC)

Amazon Virtual Private Cloud (Amazon VPC) lets you provision a logically isolated section of the Amazon Web Services (AWS) Cloud where you can launch AWS resources in a virtual network that you define. Click here to learn more about VPC.

6.2.1 Rackspace Base Network VPC

For most Fanatical Support for AWS customers, Rackspace recommends the deployment of a single VPC per AWS account to provide operational simplicity while meeting stringent security requirements. Segregation will be accomplished by creating Public and Private subnets, and by relying on carefully created Security Groups that only allow the required granular access (further details in the *Security section*).

If further segregation were required to control access (e.g. Production vs. Test vs. Development), Rackspace's recommendation is to create a separate AWS accounts, and **not** a separate VPC in the same AWS account. This is because a second VPC in the same AWS account does not provide control plane isolation of resources, and could complicate ongoing operational processes.

You can assign a single CIDR block to a VPC. The allowed block size is between a /28 netmask and /16 netmask. In other words, the VPC can contain from 16 to 65,536 IP addresses. You cannot change the size of a VPC after you have created it. If your VPC is too small to meet your needs, you will need to create a new, larger VPC, and then migrate your instances to the new VPC.

The VPC requires an RFC 1918 CIDR range (Private addresses). The default is a /16 network, however, you need to carefully consider the range you select to ensure the range does not overlap with your other environments (on premise, other AWS VPCs, Rackspace dedicated environments, etc.).

The CloudFormation template captures the CIDR range in the Parameter: VPCCIDR. Detailed IP addressing recommendations will be discussed in the *Subnets section*. The BaseNetwork CloudFormation template's default VPC CIDR is 172.18.0.0/16.

6.3 Availability Zones (AZs)

Each region contains multiple distinct locations called Availability Zones, or AZs. Each Availability Zone is engineered to be isolated from failures in other Availability Zones, and to provide inexpensive, low-latency network connectivity to other AZs in the same region.

An Availability Zone (AZ) is one or more data centers in close geographic proximity connected together over low-latency/high-speed links.

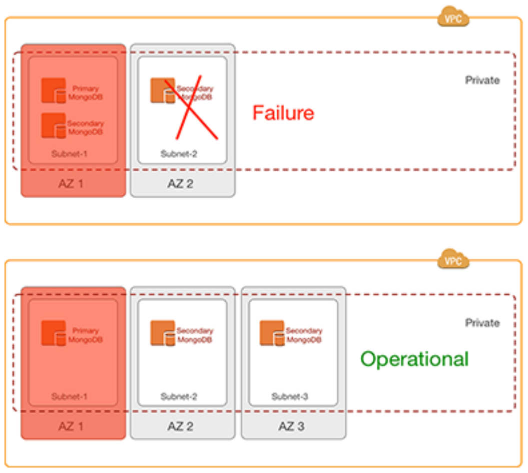
By launching instances in separate Availability Zones, you can protect your applications from the failure of a single location. Note: Each AWS region provides a minimum of two AZs.

6.3.1 Rackspace Availability Zone Recommendations

Rackspace typically recommends a two AZ deployment, which provides availability and redundancy while reducing complexity, operational overhead, and cost.

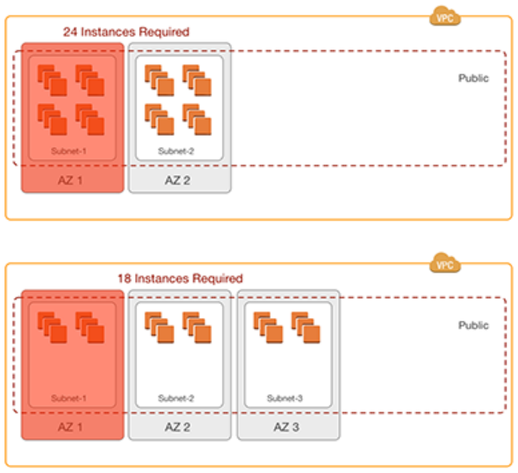
There are situations where a third AZ may be required to address specific application-centric requirements:

- Example 1: MongoDB's Election and Quorum constraints require three AZs to survive a single AZ failure that contains the primary and a secondary in a three-node cluster.



Example: If AZ 1 were to fail with only two AZs, the MongoDB cluster would fail since Election and Quorum constraints were not achieved. With three AZs, Election and Quorum constraints are achieved and the MongoDB cluster remains operational.

- Example 2: Applications that have strict load and availability requirements that cannot be met by relying on Auto Scaling Groups require over-provisioning. Adding a third AZ could be considered to reduce costs by lowering needed the over-provisioning.



Example: Strict application load and availability requirements dictates 12 servers to be up at all times, even if one AZ fails, (assuming AutoScale cannot scale fast enough during an AZ failure). This requires over provisioning. Adding more AZs to the architecture would reduce cost, but could potentially add complexity.

6.4 Subnets

You can create a VPC that spans multiple Availability Zones. After creating a VPC, you can add one or more subnets in each Availability Zone. Each subnet must reside exclusively within one Availability Zone and cannot span zones. AWS assigns a unique ID to each subnet.

If a subnet's traffic is routed to an Internet gateway, the subnet is known as a Public subnet. If the instance in a Public subnet needs to communicate with the Internet, it must have a public IP address or an Elastic IP address. If a subnet doesn't have a direct route to the Internet gateway, the subnet is known as a Private subnet.

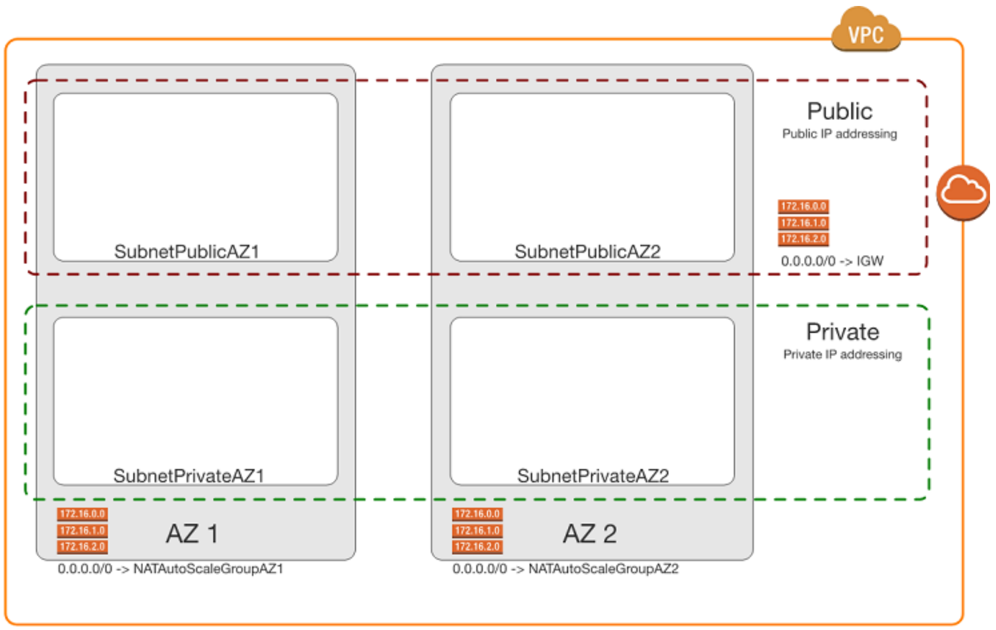
When you create a subnet, you specify the CIDR block for the subnet. The CIDR block of a subnet can be the same as the CIDR block for the VPC (for a single subnet in the VPC), or a subset (to enable multiple subnets). The allowed block size is between a /28 netmask and /16 netmask. You can create more than one subnet in a VPC, but the CIDR blocks of the subnets must not overlap.

6.4.1 Rackspace Subnet Recommendations

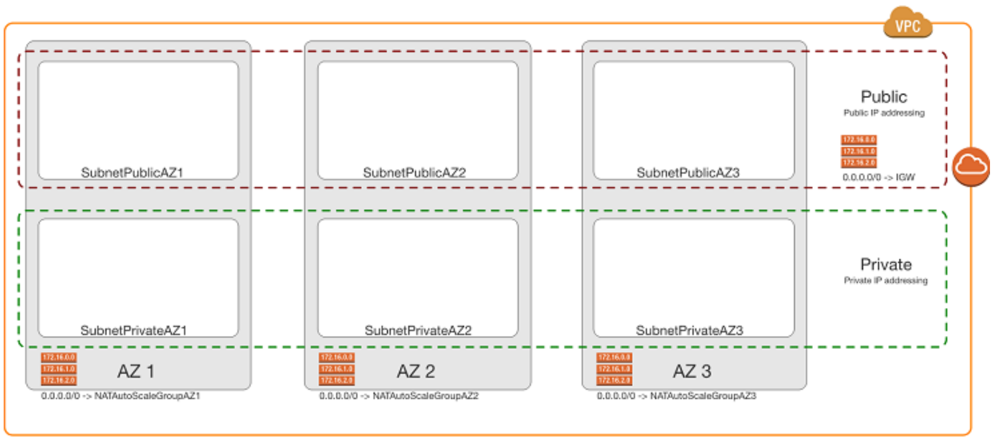
For most deployments, Rackspace recommends having two tiers of Subnets: Public and Private.

- EC2 instances in Public Subnets have public IP addresses associated with them and have a direct route to an AWS Internet Gateway (IGW), thus having the capability (if required) to access or be accessed by the Internet.
- EC2 instances in Private Subnets only have private IP addresses and cannot be accessed by the Internet. These EC2 instances have the capability to access the Internet via a NAT Gateway in the Public subnets (further info in the *NAT section*).

Assuming a typical two AZ deployment, four subnets would be required (two for Public and two for Private).



In situations where a third AZ is required (e.g. MongoDB servers in the Private subnets) then six subnets would be required (three for Public and three for Private).



It is important to note that within each tier, all the subnets will have the same network mask to simplify the operational processes (e.g. /22 for all Public subnets and /21 for all Private subnets).

Unlike traditional networking segmentation approaches that requires separate subnets (VLANs) for web, batch, application, and data tiers, AWS's use of Security Groups allows you to leverage just the Public and Private subnets, applying specific Security Groups to each tier (further info in the *Security section*). Thus a deployment would looks like:

- Public Subnets
  - Bastion servers
  - NAT servers (if not using a NAT Gateway)
  - VPN servers (if not using a Virtual Private Gateway)
  - Web servers not behind any ELB
- Private Subnets
  - Web servers behind an ELB
  - Batch-tier instances
  - App-tier instance
  - Data-tier instances

The CloudFormation template uses the built-in "GetAZs" function to map the first or second AZ to the specified subnet in a particular region (e.g. us-west-1a and us-west-1b). The CloudFormation template also captures the CIDR range for the subnet in the parameters:

- SubnetPublicAZ1 - CIDR for Public subnet
- SubnetPublicAZ2 - CIDR for Public subnet
- SubnetPrivateAZ1 - CIDR for Private subnet
- SubnetPrivateAZ2 - CIDR for Private subnet
- SubnetProtectedAZ1 - CIDR for Protected subnet
- SubnetProtectedAZ2 - CIDR for Protected subnet

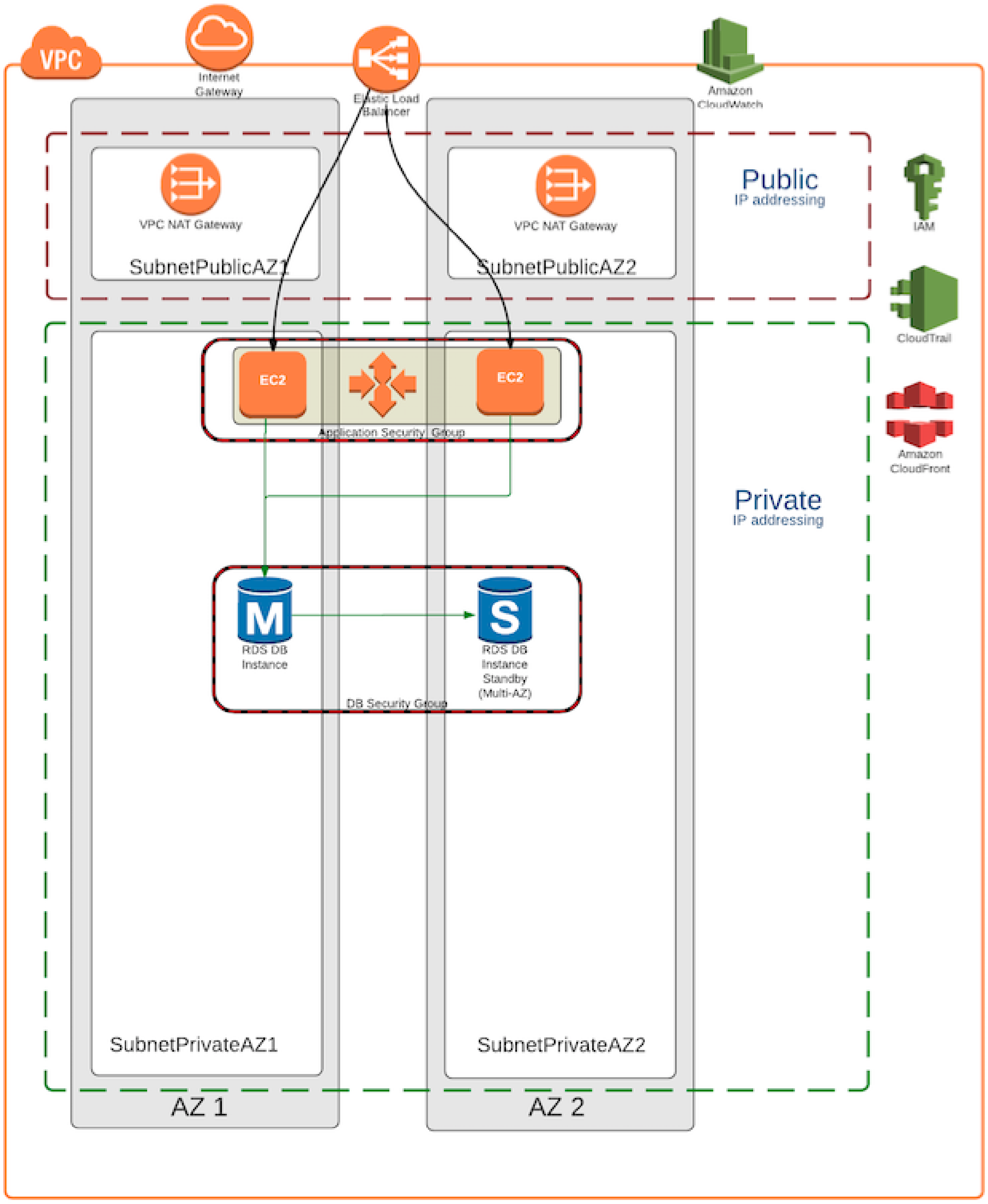
It is recommend that you choose the CIDRs carefully to map with the applications' requirements; however, most AWS customers typically allocate roughly double the IP addresses for private subnets than public subnets. The default CIDRs in the BaseNetwork CloudFormation template are detailed in the *CloudFormation section*.

6.5 Highly Available Network Address Translation (HA NAT)

In a VPC, you can use private subnets for instances that do not require directly accessible Internet-facing IP addresses. Instances in a private subnet can access the Internet without exposing their private IP address by routing their traffic through a Network Address Translation (NAT) gateway in a public subnet. Each NAT gateway is created in a specific Availability Zone (AZ) and has built-in redundancy in that AZ.

6.5.1 Rackspace NAT Recommendations

As mentioned above, NAT gateways are required for instances in a Private subnet to access the Internet. In the recommended *two AZ deployment*, Rackspace recommends leveraging one NAT gateway in each AZ - not sharing a NAT gateway with more than one AZ.



NAT gateways are created via the CloudFormation template which:

- Creates an Elastic IP Address (EIP) for each NAT gateway to be reachable on public networks
- Creates a route for each private network in each AZ to route all traffic through the corresponding NAT gateway in each AZ.

Rackspace does not recommend creating resources in one AZ that rely exclusively on a NAT gateway in a different AZ. In the event of a NAT gateway failure, resources in any AZ that depend on that single NAT gateway will be unable to access the Internet.

6.5.2 Migrating from a NAT instance

If you were previously using NAT instances for allowing resources on private networks to access the Internet, you should create a NAT gateway in each AZ, and change the routing tables for your private networks to use the new NAT gateway. Then, existing resources associated with your NAT instances (autoscale groups, NAT instances in EC2) can be removed; the change will only impact connections open at the time of the change to the routing table.

You should also take care to ensure that any existing whitelist of IPs from the NAT instances are also adjusted to reflect the new IPs of your NAT gateways.

6.6 Security

AWS provides a scalable, highly reliable platform that helps customers deploy applications and data quickly and securely.

When customers build systems on the AWS infrastructure, security responsibilities are shared between the customer and AWS. This shared model can reduce the customer's operational burden as AWS operates, manages, and controls the components from the host operating system and virtualization layer down to the physical security of the facilities in which the services operate. In turn, the customer assumes responsibility and management of the guest operating system (including updates and security patches), other associated applications, as well as the configuration of the AWS-provided security group firewall.

The Rackspace Fanatical Support for AWS offering takes some of the security burden from the customer by leveraging AWS security best practices and providing additional security capabilities. These include using/enabling Security Groups, Config, CloudTrail, CloudWatch, etc. In this section, we will focus on Security Groups.

6.6.1 Security Groups

A Security Groups acts as a virtual firewall that controls inbound and outbound traffic for one or more instances. When an instance is launched in a VPC, the instance can be assigned up to five security groups that are associated to the VPC. Specific inbound and outbound rules are then added to each security group that allows defined traffic to or from its associated instances. Rules can be modified at any time; the new rules are automatically applied to all instances that are associated with the security group.

Note: By default, outbound rules allow all traffic to egress the instance and inbound rules allow nothing (implicit deny).

Security groups act at the instance level, not the subnet level. Therefore, each instance in a subnet in a VPC could be assigned to a different set of security groups, thus easily creating isolation within the same subnet. When AWS is deciding whether to allow traffic to reach an instance, all the rules from all the security groups that are associated with the instance are evaluated at the same time. This is very different to the way Network ACLs (NACLs) work.

6.6.2 Network ACLs (NACLs)

AWS also offers network ACLs with rules similar to your security groups. NACLs act as a firewall for associated subnets, controlling both inbound and outbound traffic at the subnet level. The following summarizes the basic differences between security groups and network ACLs:

Security Group

- Operates at the instance level (first layer of defense)
- Supports allow rules only
- Is stateful: Return traffic is automatically allowed, regardless of any rules
- We evaluate all rules before deciding whether to allow traffic
- Applies to an instance only if someone specifies the security group when launching the instance, or associates the security group with the instance later on

Network ACL

- Operates at the subnet level
- Supports allow rules and deny rules
- Is stateless: Return traffic must be explicitly allowed by rules
- We process rules in number order when deciding whether to allow traffic
- Automatically applies to all instances in the subnets it's associated with (backup layer of defense, so you don't have to rely on someone specifying the security group)

6.6.3 Rackspace Security Model

As a general best practice, Rackspace advises customers to use Security Groups as their primary method of securing workloads within AWS. While Network ACLs (NACLs) are typically more familiar to networking engineers, they often introduce complexity into AWS architectures.

Security Groups provide more granular control, are stateful (therefore more intelligent in allowing appropriate traffic) and apply only to the instance level. By using NACLs as well as Security Groups, you must consider all traffic in a stateless context (specifying inbound and outbound ports, including any ephemeral ports used by a given application) and these rules are applied at a subnet level; the "blast radius" or potential for impact when a NACL is incorrect or changed is significantly higher, without providing any tangible benefit over the use of a Security Group.

Rackspace and AWS recommend avoiding NACLs due to potential conflicts with Security Groups and performance degradation. If there are compliance requirements (e.g. PCI) that specifically call for NACLs, they will be used sparingly and with coarse controls to mitigate potential issues.

6.7 Tagging

AWS customers use tags to organize their EC2 resources (instances, images, load balancers, security groups, and so forth), RDS resources (DB instances, option groups, and more), VPC resources (gateways, option sets, network ACLS, subnets, and the like), Route 53 health checks, and S3 buckets. Tags are used to label, collect, and organize resources and become increasingly important as customers use AWS in larger and more sophisticated ways.

For example, customers can tag relevant resources and then take advantage *cost allocation via tagging*.

6.7.1 Rackspace CloudFormation Tagging

The BaseNetwork CloudFormation template makes use of tagging to drive many of the operational functions associated with the Fanatical Support for AWS offering. These include:

- Service Provider - "Rackspace"

- Environment - from Parameter Environment
- Name - Resource name (e.g. IGWBase, SubnetPublicAZ2)

CHAPTER

SEVEN

BILLING

When you signup for Fanatical Support for AWS, Rackspace establishes one or more AWS accounts for you and becomes your reseller of AWS services. This means that all billing of both infrastructure and management fees is provided through a consolidated Rackspace bill, and you do not have to maintain a payment relationship with AWS directly. The credit card you provided when signing up for your Rackspace account will be automatically billed for both your AWS infrastructure and management fees, as described below.

7.1 Billing Currency

Our default pricing and billing currency is US Dollars (USD). However, when you signup, you can choose to be billed in Australian Dollars (AUD). Customers that choose the AUD billing option should be aware of the following:

- All pricing (AWS infrastructure and management fees) will remain in USD.
- The bill will be converted from USD to AUD at monthly billing processing time, using a recent spot exchange rate.
- Non-USD billing will incur a premium: 1% for customers billed using a credit card, 2% for customers billed via an invoice. This will be added to the spot exchange rate.
- The exchange rate will be displayed on the invoice.
- Compass, Waypoint, and Usage Details will show pre-converted costs in USD. The invoice will show costs in AUD.
- It is not possible to change the billing currency on existing accounts.

7.2 Billing Cycles

AWS bills for all infrastructure on a calendar month basis. AWS charges for the previous month's usage are typically finalized by the 10th day of each month. After the charges are finalized by AWS, both infrastructure and support charges are added to your Rackspace account and will appear on your next Rackspace bill. Each line item will include the month in which the charges were incurred. Your Rackspace bill is created the 15th of each month if you signed up for a Fanatical Support for AWS Account after July 6, 2017. If you signed up prior to July 6, 2017 or are using an account originally created for the Rackspace Public Cloud, you will be billed based on the anniversary date the account was created.

7.3 Financial Benefits of your Rackspace account

Your Rackspace account is the top-level container which contains one or more AWS accounts. When aggregating the usage to generate a bill at the Rackspace account level, you receive the following benefits:

- Favorable *Reserved Instance allocation*
- Tiering of usage across all accounts for AWS services which provide tiered pricing (for example, if S3 has a 0-10 TB storage tier and a 10-20 TB storage tier and you have one AWS account which uses 8 TB and another which uses 3 TB your overall usage would be rated as a combined 11 TB)

7.4 Monthly Service Fees

Your monthly service fees will be calculated by pooling the AWS infrastructure charges from all of your AWS accounts at the same service level. If the AWS infrastructure charges are greater than $30, a monthly service fee will be assessed and proportionally distributed out to each of your AWS accounts at that service level.

For example, if you have two AWS accounts at the Aviator service level, one with $45,000 in AWS infrastructure charges and the other with $30,000 in AWS infrastructure charges, your monthly service fee would be $25,000 based on your combined AWS infrastructure charge of $75,000.

During your first month with any AWS accounts at a specific service level, we will prorate the monthly service fee for the remainder of the month based on your signup date unless you make any Reserved Instance purchases during that month. If you do make a Reserved Instance purchase, we remove the proration as the service fees paid during your first month will cover our services for the length of the reserved instance reservation. Please note that proration is based on AWS account signup date, and not the date AWS infrastructure charges reach more than $30. For example, if your AWS account has a signup date of March 15th but you do not start using resources in your account until April 5th, you will be charged a full month of service fees for April.

7.5 Usage

The Usage page in the Fanatical Support for AWS Control Panel will provide you with a mid-month view of your charges and an estimate of your full month's charges, typically updated a few times per day, along with historical usage from previous months. You can use this information to avoid unexpected AWS infrastructure charges. To access the report, select the Usage link in the primary navigation.

Due to our *Account Defaults* and associated management tooling, each AWS account that you provision will have approximately $5-10 of monthly infrastructure charges, regardless of whether you provision any additional AWS resources.

7.6 Viewing your Invoices

To view your invoices, log in to the Fanatical Support for AWS Control Panel and click the Billing link toward the top right of the page.

The primary account holder will receive an email any time a payment is processed, indicating that a new invoice is available for review.

7.7 Tagging

Rackspace will provide detailed views of your AWS billing data by resource tags. Tags must use a key from the following list (case sensitive) in order to be included in these views:

- BusinessUnit
- Group
- Department
- CostCenter
- Application
- Environment
- Project
- Owner
- Service
- Cluster
- Role
- Customer
- Version
- Billing1
- Billing2
- Billing3
- Billing4
- Billing5

We also include the following AWS-generated tags in the detailed views of your AWS billing data:

- aws:autoscaling:groupName
- aws:cloudformation:logical-id
- aws:cloudformation:stack-id
- aws:cloudformation:stack-name

While you may use tags outside of those listed above to identify your resources for other reasons, they will not be included in the detailed views of your billing data.

7.8 Modifying your Payment Method

If you need to update the credit card or ACH (eCheck - United States only details that you have on file, log in to the Fanatical Support for AWS Control Panel and click the Billing link toward the top right of the page. From there, you'll find a link to update your payment details.

CHAPTER

EIGHT

RESERVED INSTANCES

Reserved Instances play an important role in helping you manage the overall costs of your AWS environments. In cases where you plan to have sustained 24x7 usage of one or more EC2 or RDS instances of the same instance type in the same availability zone and region, purchasing a one-year or three-year reserved instance can save up to 70% versus the on-demand hourly rates.

You can learn more about Reserved Instances at https://aws.amazon.com/ec2/pricing/reserved-instances/.

8.1 Allocation across AWS accounts

If you have more than one AWS account that is part of the same Rackspace account you can benefit from automatic allocation of unused reserved instances from one AWS account to another. Our billing system automatically detects unused reserved instances on your AWS account and searches for corresponding EC2/RDS on-demand instances of the same instance type and provisioned in the same availability zone on your other AWS accounts under the same Rackspace account. If a match is found, the reserved instance is automatically applied to the usage on the other AWS account.

A few key considerations:

- Reserved instances are a billing construct only; you do not need to specify anything at the time you launch the EC2 or RDS instance. As long as the instance type and availability zone match, the reserved instance benefit will automatically be applied.
- The allocation of reserved instances to other accounts occurs on an hourly basis; therefore, if the account that purchased the reserved instance originally begins using the same instance type in the same availability zone at a later time, the reserved instance benefit will be applied to the original purchasing AWS account.
- Although the underlying data centers powering a specific availability zone vary across accounts (e.g. us-east-1a may be different on account X than account Y), for the purposes of reserved instance allocation the availability zone match is done based only on the availability zone name.

8.2 Purchasing Reserved Instances

You can purchase reserved instances directly from AWS via the AWS Console or CLI, or programmatically via the SDKs or CLI.

8.3 Impact on Monthly Service Fees

As described in the *Pricing* section, your monthly service fee is calculated based on the total of all AWS infrastructure charges. Reserved instance purchases are included in these charges, so months where you make one or more reserved instance purchases may cause you to incur a higher monthly service fee. Since the reserved instance will lower or eliminate your charges for that portion of the infrastructure in future months, you'll see a similar benefit apply to your monthly service fees. The effective rate of our service fees decreases as AWS infrastructure spend increases, so you will likely pay a lower total amount of service fees over the life of the reserved instance than if you had run on-demand instances during the same time period.

8.3.1 Example 1: Navigator Service Level

For this example, assume that your application requires a single m4.4xlarge instance and no other AWS on-demand infrastructure. Your options are:

- Reserved Instance Purchase
  - Reserved Instance Purchases - month 1 (one m4.4xlarge instance; one year; all up front): $5,082
  - Service fee - month 1 (based on spend of $5,082): $750
  - Service fee - months 2 through 12 (based on spend of $0): $0
  - Total cost: $5,832
- No Reserved Instance Purchase
  - On-demand usage - months 1 through 12 (one m4.4xlarge instance): $735.84
  - Service fee - months 1 through 12 (based on spend of $735.84): $400
  - Total cost: $13,630.08

In the example above, you would save 84% on service fees and 42% on AWS infrastructure costs when purchasing reserved instances.

8.3.2 Example 2: Aviator Service Level

For this example, assume that your application requires 25 c4.2xlarge instances and no other AWS on-demand infrastructure. Your options are:

- Reserved Instance Purchase
  - Reserved Instance Purchases - month 1 (25 c4.2xlarge instances; one year; partial front): $33,648.25
  - Service fee - month 1 (based on spend of $33,648.25): $17,000
  - Reserved Instance monthly fees - months 2 through 12 (25 c4.2xlarge instances): $2,573.25
  - Service fee - months 2 through 12 (based on spend of $2,573.25): $2,500
  - Total cost: $106,454
- No Reserved Instance Purchase
  - On-demand usage - months 1 through 12 (25 c4.2xlarge instance): $8,048.25
  - Service fee - months 1 through 12 (based on spend of $8,048.25): $5,750
  - Total cost: $165,579

In the example above, you would save 36% on service fees and 36% on AWS infrastructure costs when purchasing reserved instances.

8.3.3 Example 3: Navigator Service Level

For this example, assume that your application requires 15 c4.2xlarge instances and no other AWS on-demand infrastructure. Your options are:

- Reserved Instance Purchase
  - Reserved Instance Purchases - month 1 (15 c4.2xlarge instances; one year; partial front): $20,188.95
  - Service fee - month 1 (based on spend of $20,188.95): $3,500
  - Reserved Instance monthly fees - months 2 through 12 (15 c4.2xlarge instances): $1,543.95
  - Service fee - months 2 through 12 (based on spend of $1,543.95): $750
  - Total cost: $48,922.40
- No Reserved Instance Purchase
  - On-demand usage - months 1 through 12 (15 c4.2xlarge instance): $4,828.95
  - Service fee - months 1 through 12 (based on spend of $4,828.95): $1,500
  - Total cost: $75,947.40

In the example above, you would save 35% on service fees and 36% on AWS infrastructure costs when purchasing reserved instances.

Note: The costs used in the examples above are for illustrative purposes only and could change at any time. In some cases, your service fees will be higher when purchasing reserved instances versus not purchasing them, decreasing, but not eliminating, your overall savings. Your *Account Manager* can assist you with calculating reserved instance benefits for your specific account.

8.4 Additional Billing Information

For detailed billing information and methodology related to Reserved Instances, please reference the AWS documentation at https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts-reserved-instances-application.html.

CHAPTER

NINE

ACCESS AND PERMISSIONS

Controlling access and permissions to the Rackspace and AWS control planes (APIs and UIs) along with the resources you deploy at AWS are a critical part of the overall security of your environment. This section outlines several core concepts related to access and permissions, along with details on how to grant members of your team and others access to your account, as needed.

9.1 User Management and Permissions

9.1.1 Account Owner

When you sign up for Fanatical Support for AWS, the first user you create is the **Account Owner**. After signing up, you can reassign Account Owner status to another user on the account. You can make this change from the **Account Settings** page. There can only be one Account Owner at a time.

The **Account Owner** has full administrative privileges, including:

- AWS `AdministratorAccess` IAM policy rights on all AWS accounts
- `Admin` rights to all Fanatical Support for AWS features on all AWS accounts
- `Admin` rights to the Rackspace Billing and Payments portal
- Ability to add additional AWS accounts for Rackspace to manage
- Ability to create and delete users on the Rackspace account and manage their permissions on a per AWS account basis
- Ability to make other users Account Administrators
- Ability to reassign Account Owner status
- Ability to configure Rackspace account-wide settings including enabling multi-factor authentication, configuring session duration, etc.
- Ability to cancel the Rackspace account

9.1.2 Creating and Managing Users

If you have more than one person in your organization that will need access to the Fanatical Support for AWS Control Panel, the AWS Console/APIs, or both, you can create additional users and assign them permissions on a per AWS account basis.

To create and manage users:

1. Log in to the Fanatical Support for AWS Control Panel.

2. Click your **user name** at the top right to activate the account menu.
3. Select **User Management**.

From the **User Management** page you'll have the ability to create new users, manage existing users, and assign permissions to users.

9.1.3 Identity Federation

Rackspace Identity Federation enables you to configure your corporate security and identity systems to enable your employees to use their regular company credentials to authenticate to Rackspace accounts. For more information about Identity Federation, see the Rackspace Identity Federation User Guide.

9.1.4 Account Administrator

One permission that can be assigned is **Account Administrator**. This permission is useful when the Account Owner wishes to delegate a significant set of rights to another user on the account, e.g. if someone else in the organization besides the Account Owner is responsible for creating new users and assigning them permissions.

Users with the **Account Administrator** right have the following privileges:

- AWS `AdministratorAccess` IAM policy rights on all AWS accounts
- `Admin` rights to all Fanatical Support for AWS features on all AWS accounts
- `Admin` rights to the Rackspace Billing and Payments portal
- Ability to add additional AWS accounts for Rackspace to manage
- Ability to create and delete users on the Rackspace account and manage their permissions on a per AWS account basis

Account Administrators do **NOT** have the following permissions:

- Ability to view or modify the Account Owner or other users with the Account Administrator permission
- Ability to make other users Account Administrators
- Ability to configure Rackspace account-wide settings including enabling multi-factor authentication, configuring session duration, etc.
- Ability to cancel the Rackspace account

The **Account Administrator** permission does not determine user status within CloudHealth. Please see **Account Owner** and the Product Permissions section for CloudHealth user status controls.

9.1.5 Understanding and Managing Permissions

The **Account Owner** and **Account Administrators** have the ability to manage permissions for other users.

There are two categories of permissions:

1. Rackspace Account Permissions

These permissions are Rackspace account-wide and broader than Fanatical Support for AWS.

- **Account Administrator** - Gives the user a substantial subset of Account Owner permissions (see above for details).

- **Billing and Payments** - Provides access to the Rackspace Billing and Payments portal which includes information like invoices, payment methods, and billing settings.
- **Support Tickets** - Provides the ability to give more granular access to your Rackspace Account support tickets. You can prevent users from seeing tickets. You can also allow users to only see tickets, however, they will not be able to create tickets.

2. Product Permissions

These permissions are Rackspace product specific. This is where Fanatical Support for AWS permissions are managed (other product permissions will not be covered in this guide).

There are three Fanatical Support for AWS permissions:

- **Allow this user to add AWS Accounts** - Enables the user to add additional AWS accounts for Rackspace to manage. These could be new or existing AWS accounts.
- **Fanatical Support for AWS** - Controls what access, if any, the user will have within the Fanatical Support for AWS Control Panel. This permission applies to **all** Rackspace features including Passport, Logbook, Compass, and Usage. This permission is configured on a per AWS account basis.
- **AWS Console and APIs** - Controls what access, if any, the user will have when federating to the AWS Console or retrieving AWS temporary API credentials. This permission can be any AWS managed or custom IAM policy available on the AWS account and is configured on a per AWS account basis.

Rackspace Permission Types

Rackspace specific permissions can be set to one of three values:

- **None** - No access
- **Observer** - Read-only access
- **Admin** - Read and write access

9.1.6 Permission Example

You have two AWS accounts managed by Rackspace, both at the Aviator service level. They are named **App1-Staging** and **App1-Production**.

You might grant a junior developer working on this application the following permissions:

Account Permissions

- **Account Administrator** - `Disabled`
- **Billing and Payments** - `None` since he does not need access to invoice and payment information

Product Permissions

- **Allow this user to add AWS Accounts** - `Disabled`

App1-Staging

- `Admin` access to **Fanatical Support for AWS** so, for example, he has the ability to authenticate to instances via Passport.

- `AdministratorAccess` IAM policy access so he has full access to AWS services via the **AWS Console and APIs**.

App1-Production

- `Observer` access to **Fanatical Support for AWS** so he can view but not make changes to the production AWS Account via Rackspace tooling. This will disable Passport access but Compass and Logbook are still available.
- `ViewOnlyAccess` IAM policy to limit his **AWS Console and API** access to view-only.

CloudHealth Permissions

CloudHealth views are available at a default organization or sub-organization level. The default organization is a view of all AWS accounts under your Rackspace account. The sub-organization view is only the individual AWS accounts for which you have AWS control plane access.

CloudHealth access is given at either a Power User or Standard User permission. A Power User has full operational privileges across all data. A Standard User can view but not edit or delete data within CloudHealth.

Rackspace Role	CloudHealth User Status	CloudHealth Organization
Account Owner	Power User	Default Org
Product Access: Admin	Power User	Default Org
Product Access: Observer	Standard User	Default Org
Fanatical Support for AWS: Admin	Power User	Sub-Org
Fanatical Support for AWS: Observer	Standard User	Sub-Org

Please contact your Rackspace Customer Success Manager with any questions about permissions.

9.2 Rackspace Account

Your Rackspace account is the top-level container which contains one or more AWS accounts. All user and permissions management takes place at the Rackspace account level, though you can limit specific users on your account to only have access to specific AWS accounts. The Rackspace account is also used for billing purposes. All charges from each of the AWS accounts are *aggregated at the Rackspace account level*.

The Account Owner or a user with the Account Administrator designation can set the session inactivity timeout for the Fanatical Support for AWS Control Panel. To change the timeout, visit account.rackspace.com, and then navigate to Account Settings > Rackspace Account Settings > Session Inactivity Timeout. The session inactivity timeout is the maximum time a user can be inactive before being automatically logged out. This timeout is applied to all users on the account.

9.3 AWS Console

Once you have logged in to the Fanatical Support for AWS Control Panel you will see a listing of all AWS accounts you have access to.

If you wish to access the AWS Console you can click the “Log in to AWS Console” button and you will be automatically signed in as a federated user. This allows you to maintain one set of credentials to access both the Fanatical Support for AWS Control Panel and the AWS Console. As described in the *User Management and Permissions section* the access a user will receive when they federate to the AWS Console will be determined by the AWS IAM Role selected when configuring the user’s permissions.

The Account Owner can set the AWS console session duration, which controls the maximum session time for a user who logs into the AWS console from the Fanatical Support for AWS Control Panel. To change the timeout, visit account.rackspace.com, and then navigate to Account Settings > Product Preferences > AWS Account Preferences > AWS Account - AWS Console Session Duration. This maximum limit is applied to all users on the account.

9.4 AWS CLI, SDKs, and APIs

There are two methods for accessing the AWS command-line interface (CLI), software development kits (SDKs), and application programming interfaces (APIs):

1. From the Fanatical Support for AWS Control Panel, navigate to the Account Details screen for the AWS account you would like to access and click the View Credentials button. You will be issued AWS Security Token Service (STS) credentials that are valid for up to 60 minutes and are scoped to the same level of permissions as if you were to federate to the AWS Console. This is the preferred method of short-lived, infrequent access as access to the credentials is tied to your Fanatical Support for AWS user and is logged in the *Rackspace Logbook*.
2. If you require longer-lived, more persistent access to the CLI, SDKs, or APIs you should create an IAM user with access keys (if the access will be from a user's workstation) or an IAM instance role (if the access will be from resources, such as EC2 instances, running at AWS). Note that directly-created IAM users or roles are not managed within the Fanatical Support for AWS user management system, and therefore modifying or terminating access must be done directly within AWS IAM.

If you need assistance determining which option is best for your specific use case, please *contact a Racker*.

9.5 AWS Identity and Access Management (IAM)

As described earlier, our standard best practice is to manage all access as either:

- Users within the Fanatical Support for AWS Control Panel
- IAM Roles for AWS resources, such as EC2 instances, requiring access to other AWS services

Occasionally, a use case will arise where it is necessary to directly create an IAM user or role. These scenarios typically involve a third-party tool or SaaS needing access to your account, such as a continuous integration and deployment system like CircleCI or a local file management application that integrates with S3 such as Cyberduck. If you must create a user or role directly within IAM, please remember the following:

- The IAM policy that you assign should be created to allow the minimum level of access required to your AWS account. If you need assistance with creating the appropriate IAM policy, please *contact us*.
- IAM users and roles are managed outside of the Fanatical Support for AWS Control Panel and will not show up in the User Management system. Therefore, any modifications or revocation of access must also be performed directly within AWS IAM.
- A default IAM password policy is included in our *AWS account defaults*. We do not recommend weakening or disabling these requirements, as they are put in place to protect your account from brute-force password attacks.
- An IAM user should typically have password access or access keys, but not both. Password access is used for accessing the AWS Console (and most of these use cases should be covered under the Fanatical Support for AWS Control Panel permissions model) and access keys are used for programmatic access. In almost all cases where you are creating an IAM user, only access keys should be required.

For assistance in determining the appropriate method of granting access to your account, please *contact us*.

9.6 AWS Systems Manager EC2 Session Manager

AWS accounts managed by Rackspace require the use of the AWS Systems Manager Agent for operating system support.

AWS Systems Manager Session Manager may be used to provide shell access to Operating Systems via the AWS console or CLI. You can learn more about Session Manager at: ‘<https://docs.aws.amazon.com/systems-manager/latest/userguide/session-manager.html>’_.

It should be noted that use of AWS Session Manager will result in commands being executed under a shared user account (ssm-user) within the Operating System. This user will persist even if the SSM agent is removed. Customers with specific compliance or internal security policies should consult with their compliance personnel on whether Session Manager is appropriate for their use. For compliance information from AWS, see: ‘<https://aws.amazon.com/compliance/services-in-scope/>’_.

Rackspace recommends customers secure their Rackspace and AWS accounts with 2-factor authentication. Customers may restrict their personnel’s usage of AWS Session Manager via AWS IAM.

Rackspace personnel may use Session Manager as needed to perform administrative tasks. :ref:’account_defaults’ ensure any Rackspace usage of AWS Systems Manager and its associated features is logged in AWS CloudTrail.

CHAPTER

TEN

SECURITY

10.1 Rackspace Shared Management Services

Rackspace takes the security of our shared management services and the Fanatical Support for AWS Control Panel extremely seriously. All infrastructure is deployed on AWS leveraging the same set of best practices that we apply to customer accounts. The following sections provide a sample of some of the key security focus areas.

10.1.1 Racker Authentication

All Rackspace employees must leverage two-factor authentication for all access to customer account data and customer environments.

10.1.2 Racker Privileges

The level of privileges each Racker has to our Fanatical Support for AWS management systems is tightly controlled based on job role and is periodically reviewed to ensure that each Racker has the minimum level of permissions required to adequately perform their job duties. All privilege changes require management approval and are also logged for later review.

10.1.3 Encryption at Rest

All databases leverage the AWS Key Management Service (KMS) for data encryption at rest. All EBS volumes are encrypted with KMS in addition to application-level encryption of secrets using KMS and the AWS SDKs.

10.1.4 Encryption in Transit

All communication between services that make up the Fanatical Support for AWS shared management system are encrypted during transit using SSL. Our customer and Racker UIs and APIs are only accessible via HTTPS.

10.1.5 AWS Account Best Practices

As outlined in the *AWS Accounts section* we always enable AWS CloudTrail and AWS Config in all regions for each new account. We also have checks within Rackspace Compass that ensure these remain enabled and configured per our best practices.

10.1.6 Activity Logging

As described in the *Rackspace Logbook section* all control plane and data plane activities are logged and visible to both customers and Rackers via the Fanatical Support for AWS Control Panel, providing a complete playback of events that occurred on an account.

10.2 AWS Security

Amazon Web Services places a high degree of importance on the security of your infrastructure. For an overview of the AWS Security Processes we recommend reviewing their whitepaper.

We also encourage you to review the Securing Data at Rest with Encryption whitepaper for an overview of the methods for securing your data.

If you have questions regarding any aspect of the whitepapers or the security of your environment, please *contact a member of your Support team*.

10.2.1 Security Updates

As Amazon Web Services says on their Security Bulletins web page, "No matter how carefully engineered the services are, from time to time it may be necessary to notify customers of security and privacy events with AWS services."

If you are interested in staying informed about these security bulletins, the AWS Security Bulletins web page or the companion RSS feed are the appropriate channels to watch. From time to time, Rackspace will provide additional detail or guidance to our customers for specific security incidents. We will publish such value-add security updates below.

Rackspace Response to Meltdown and Spectre

On 3 January 2018, Rackspace was made aware of a vulnerability affecting certain processors by Intel, AMD and ARM. Multiple vendors have subsequently released statements regarding the vulnerability and its impact on their respective environments.

These issues were originally uncovered by Google's Project Zero. Their research findings show that an unauthorized party may read sensitive information in the system's memory such as passwords, encryption keys, or sensitive information open in applications.

The remainder of this update is addressed to our Fanatical Support for AWS customers specifically. For updates about our other Rackspace supported hosting environments, please refer to the Rackspace blog.

Overview

Details about the security vulnerabilities can be found in CVE-2017-5753, CVE-2017-5715, and CVE-2017-5754. Amazon's security bulletin regarding these vulnerabilities can be found here. The website spectreattack.com provides a good overview of these vulnerabilities.

There is not a single fix

There is no single fix for all three security vulnerabilities. Many vendors have patches available for one or more of these attacks.

Amazon Web Services (AWS) Response

These vulnerabilities affect many CPUs, including those used by Amazon EC2. As of 2018/01/04 15:30 PST, Amazon reports that "all instances across the Amazon EC2 fleet are protected. . . against these threats from other instances."

This is an important first step in mitigating the security risk that these vulnerabilities represent for your environments. As a result of this fix to the EC2 infrastructure, attackers will no longer be able to exploit EC2 infrastructure to view the contents of memory allocated to a different virtual machine running on the same hypervisor as their attacking software.

Actions that customers are responsible for taking

Despite the actions taken by Amazon to patch EC2 infrastructure, a risk remains that malicious software running within your guest operating system may be able to exploit these security vulnerabilities to gain access to private data stored in memory on your EC2 instances. To protect against this risk, Rackspace will be communicating with you over the next several days with guidance and options for patching your EC2 instances.

Watch this page for updates

Please check back at this web page for updates on this security issue. Rackspace will post updates as additional details become available from affected vendors, and we will add additional guidance to this page over the next several days.

Change Log

Date	Description
2018/01/04 18:03 CST	Initial revision of this security update
2018/01/05 08:02 CST	Update title; Update status to reflect that Amazon have completed EC2 infrastructure patching
2018/01/04 14:57 CST	Minor update to wording to better align to Amazon's security bulletin

CHAPTER

ELEVEN

COMPLIANCE

11.1 PCI-DSS

Is Fanatical Support for AWS PCI-DSS compliant?

Rackspace is a certified Level 1 Payment Card Industry (PCI) Service Provider on Fanatical Support for AWS.

What was the scope of the PCI-DSS assessment?

For Fanatical Support for AWS, the Rackspace Service Provider assessment scope is detailed in the Executive Summary document provided to all customers. This assessment includes tooling and infrastructure operated by Rackspace and excludes AWS infrastructure, which is covered under their Report on Compliance.

Fanatical Support for AWS and related systems and tooling are PCI-DSS compliant. Does that mean that my solution will be compliant as well?

Hosting a solution with Rackspace does not make a customer PCI-DSS compliant. Fanatical Support for AWS Solution Architects are happy to assist our customers in navigating our product portfolio to identify solutions which meet their regulatory needs.

Can Rackspace help my solution become PCI-DSS compliant?

Rackspace is not a Qualified Security Assessor (QSA) and therefore cannot give a qualified opinion on the PCI-DSS compliance status of a customer's solution. In addition, due to many variations in our service delivery configurations we cannot offer PCI-DSS compliant solutions "out of the box." However, Rackspace can provide services, products and an extensive partner network that will satisfy many of the necessary PCI-DSS requirements. For a detailed list of controls and how Rackspace can assist, please request the PCI Responsibility Matrix.

Can Rackspace provide proof of its PCI-DSS compliance?

Rackspace can provide the following PCI-DSS Compliance Package:

- PCI Responsibility Matrix
- PCI-DSS Report on Compliance Executive Summary
- List of controls that belong to the Service Provider
- The Rackspace Attestation of Compliance

Note: Rackspace cannot release the full PCI-DSS Report on Compliance as it contains proprietary and commercially sensitive details of Rackspace security processes.

How can I get the PCI DSS Compliance Package?

Customers can access attestation of compliance forms in the Fanatical Support for AWS control panel: under the account drop-down in the upper right-hand corner, select "Documents and Forms", and navigate to the "Rackspace Cloud Security Documents" section.

Does Fanatical Support for AWS service level matter for PCI-DSS?

Both our Navigator and Aviator customers can leverage our PCI-compliant tooling and infrastructure. Note, however, that the Aviator service level provides a greater number of value-added services to include design, service selection, monitoring, and more that make achieving PCI-DSS compliance easier.

I have general questions about Rackspace Security beyond the scope of PCI-DSS - where can I get answers to those questions?

We have a Rackspace Information Security FAQ which includes additional information around security policy, internal organization, human resources, access controls, and more. Similar to the PCI-DSS Compliance Package, it can be requested from your Fanatical Support for AWS Technical Account Manager.

11.2 HIPAA

Can Fanatical Support for AWS support HIPAA workloads?

Yes, Rackspace can act as a business associate to support customers with HIPAA workloads at AWS.

Why is it important to have a Managed Service Provider (MSP) who can manage workloads on top of AWS if AWS already provides HIPAA-eligible services?

Any business that has needs to store, process, or transmit HIPAA data needs to ensure that the Managed Service Provider they choose on top of AWS has practices in place to allow them to comply with HIPAA, as well as a signed BAA (Business Associate Agreement).

Does Fanatical Support for AWS service level matter for HIPAA?

We provide management for both Navigator and Aviator customers running HIPAA workloads at AWS. However, with Aviator service level customers can take advantage of value-add services like best-practice architecture, service selection, patching, monitoring, and ongoing operations that may make achieving HIPAA compliance easier for customers.

Do I need to maintain a Business Associate Agreement (BAA) with both Rackspace and AWS?

For the AWS accounts that Rackspace supports, you only need to sign a BAA with Rackspace.

How can I get a copy of the Fanatical Support for AWS BAA?

Please get in touch with your Fanatical Support for AWS Technical Account Manager (TAM) or Rackspace Sales Representative who can get you a copy of the Fanatical Support for AWS BAA.

CHAPTER

TWELVE

PASSPORT

The Fanatical Support for AWS offering includes access to our service at the *Aviator service level*. This is the same capability that Rackers use to access your environment. Passport leverages AWS Systems Manager to provision short lived users onto your EC2 instances and provide network access into your VPC.

Passport v2 offers several improvements over our original Passport tool, including:

- User accounts are created on demand and cleaned up after use
- Public subnets and bastion hosts are no longer required in customer VPCs
- EC2 instances with multiple Elastic Network Interfaces (ENIs) are now supported

Passport's primary concept is an **Access Request**. Each access request defines who is accessing your account, which specific EC2 instances they are accessing, the duration of the access request, and the reason for the access. Access requests default to expiring after 1 hour but can be extended up to 12 hours.

As an example, a Racker receiving a CloudWatch monitoring alarm for CPU utilization on your application server might create an access request referencing the alert ticket and granting them access to your active and passive database instances. Once troubleshooting and remediation is complete, the Racker completes the access request, immediately removing the short-lived user from your instances.

All access request actions, from access request creation through expiration, are logged in *Logbook*.

12.1 Installation

12.1.1 AWS Systems Manager Agent

All EC2 instances must have AWS Systems Manager Agent 2.3.672.0 or higher installed to work with Passport. Each EC2 instance must also be **'configured with an instance profile <https://docs.aws.amazon.com/systems-manager/latest/userguide/setup-instance-profile.html'_** that allows AWS Systems Manager to perform actions on your instances. Rackspace recommends using the `AmazonSSMManagedInstanceCORE` managed IAM policy.

You can learn more about how AWS Systems Manager is used in the *Architecture section*.

12.1.2 Passport CLI

You must install the Passport CLI on your local workstation in order to connect to instances with Passport. Before you install the Passport CLI, make sure that you have the following dependencies installed:

- AWS CLI v1.16.220 or higher
- AWS Session Manager Plugin v1.1.26 or higher

12.1.3 Linux and MacOS

Run the following command in your terminal:

```
curl -sL https://passport-packages.manage.rackspace.com/installer/latest | sudo sh -s
```

If you'd like to install the CLI directly rather than using the installation script, you can download the latest binary using the following links:

- Linux - Binary, SHA256 Checksum
- MacOS - Binary, SHA256 Checksum

Once you have installed the Passport CLI, you can verify that it is installed and working correctly by running the following command:

```
passport --help
```

12.1.4 Windows

Download and run the installer by using the following links:

- Win32
- Win64

Once the AWS Systems Manager Agent and Passport CLI are install, you're ready to start using Passport. You can learn more about how to use the Passport CLI in the *CLI Usage <cli_usage>* section.

12.2 CLI Usage

Once the Passport CLI is installed, you can log in to the CLI. The following command will open your browser and you will be prompted to log in with your Rackspace credentials:

```
passport auth login
```

Next, browse to the Passport section of the Fanatical Support for AWS control panel. Click **Create Access Request** and complete the form to initiate access to your instances.

Once the access request is active, you can connect to any instance that belongs to that access request by using the following command:

```
passport connect <instance-id>
```

When connecting to an EC2 instance running Linux, the `connect` command will automatically open an SSH connection to the target server.

When connecting to an EC2 instance running Windows, the `connect` command will provide you with connection information for an RDP connection. Here is a sample output for an RDP connection:

```
Windows Login Credentials:

Host: localhost:58829
Username: <username>
Password: <password
```

To connect with RDP, open your preferred RDP client and use the information above to connect.

12.2.1 SSH Integration

Most users will only interact with the `passport` CLI tool directly. However, you can use Passport with any SSH-based tooling by using the `ssh-config` command. This command generates an SSH config that can be used with any tooling that supports SSH config files, including `ssh` port forwarding, `scp`, and Ansible.

```
# Outputs a path to an SSH config
passport ssh-config
```

12.2.2 Forwarding Ports

SSH port forwarding is commonly used to access a service that is not directly accessible to the end user. For example, RDS database instances do not have the SSM agent installed and cannot be used with Passport directly. However, SSH port forwarding can be used to access the RDS instance by using a Passport-enabled server as an intermediary.

```
# Forwarding localhost:13306 to an RDS instance on port 3306 through i-123456
ssh -F $(passport ssh-config) -L 13306:my_instance.us-east-1.rds.amazonaws.com:3306 i-
↪123456
```

Once the above command is successfully running, you can use familiar local tools and connect them to `localhost:13306` to work with your RDS instance.

Note: You must also have security group rules in place that permit access from the intermediate instance to the AWS resource that you're accessing.

12.2.3 Copying Files

`scp` can be used to copy files to and from a target instance. The following command copies the file **data.csv** from your local workstation to your home directory on the target server **i-1234567890**:

```
# Copy data.csv from the local workstation to i-1234567890
scp -F $(passport ssh-config) data.csv i-1234567890:~
```

12.3 Permissions

Note: This behavior applies specifically to Passport v2 and represents a change in how permissions work. The original Passport tool for AWS did not require an IAM policy to use Passport.

Passport relies on two settings to determine which instances a user can access with Passport. First, the user must have their Fanatical Support for AWS permission for an AWS account set to "Admin". If the permission is set to either "Observer" or "None", the user will not have access to any instances in that AWS account. Second, the user must have an IAM policy that allows the `ssm:StartSession` action to be performed against the target instance. This setting allows you to choose which instances within an AWS account a user will have access to. Together, these two settings give you granular control over which instances can be accessed using Passport.

There are several IAM policies available out of the box that are commonly used with Passport:

- **AdministratorAccess** grants full access to all AWS resources and will allow a user to access any instance in the account using Passport.
- **PowerUserAccess** grants access to non-IAM AWS resources. It will allow a user to access any instance in the account using Passport.
- **AmazonSSMFullAccess** grants access to all actions and resources within AWS Systems Manager. It will allow a user to access any instance in the account using Passport.
- **RackspacePassportOnly** grants *ssm:StartSession* on all instances. It will allow a user to access any instance in the account using Passport, but the user won't have any other permissions in the AWS console.
- **RackspaceReadOnlyWithPassport** grants read-only access to most AWS resources as well as `ssm:StartSession` on all instances. It will allow a user to access any instance in the account using Passport.

Additionally, you can define custom IAM policies that allow you to further restrict which instances can be accessed within an AWS account. See Additional Sample IAM Policies for Session Manager for more information on how to use IAM policies to limit access to certain instances.

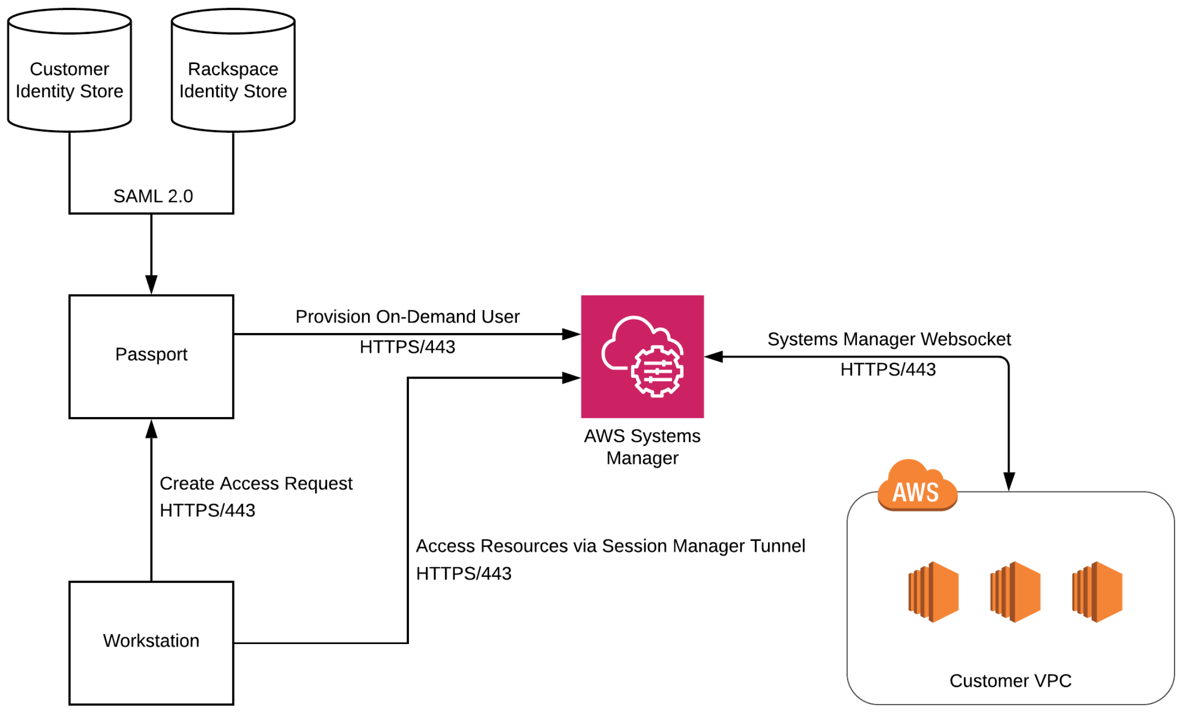
12.4 Architecture

Passport provides secure, auditable access to servers inside a Virtual Private Network (VPC). In addition to Passport, there are several other components that play a part in this feature: AWS Systems Manager and Identity Stores. This document discusses each of these components and the role they play in governing server access.

Passport handles all orchestration for an access request including: authenticating a user with their Identity Store via SAML, creation/deletion of short-lived users, and generation of temporary credentials.

AWS Systems Manager is used to create short-lived users and distribute credentials to servers that are accessed via Passport. Session Manager tunneling is also used to make a network connection to the target servers. Users are created at the time an access request is created and are cleaned up when the access request is complete. Credentials are only valid for the lifetime of the access request.

Identity Store(s) are the source of truth for all users and their associated permissions. Rackspace Customer Identity is the identity store that is used for customer users, and it can optionally be connected via SAML to an on-premise identity store via Rackspace Identity Federation. Racker permissions are stored in Rackspace's internal identity store.



1. **User creates an access request via Passport.** The user specifies their reason for access and the servers that can be accessed.
2. **Passport provisions a short-lived user on the servers being accessed.** On Linux-based systems, the short-lived user is authenticated with an SSH certificate. On Windows systems, the short-lived user is authenticated with a password. Both the SSH certificate and the password are generated by Passport and are only valid for the duration of the access request. Provisioning of the short-lived user is performed using Systems Manager Run Command.
3. **User connects to resources using a Session Manager tunnel.** Both SSH and RDP traffic are routed from the user's workstation through a Session Manager tunnel to the target resources. Once the tunnel is established, the SSH or RDP connection is authenticated using the temporary credentials generated during step 2.
4. **Passport removes the short-lived user from the server being accessed.** The short-lived user is removed and all associated credentials are revoked. Clean up of the short-lived user is performed using Systems Manager Run Command.

CHAPTER

THIRTEEN

LOGBOOK

AWS and Rackspace generate detailed control plane logs for all activities taking place in your Fanatical Support for AWS account(s). This data is aggregated from a number of different sources:

- AWS CloudTrail: detailed logs of most AWS API requests made on your account to supported AWS services. Most object-level S3 actions will be ignored. Only `PutObjectAcl`, `DeleteObject`, and `DeleteObjects` will be stored in Logbook. Read-only events are not stored in Logbook.
- Fanatical Support for AWS shared management system and user interfaces: view control panel logins and other actions (such as creating a new AWS account or modifying user permissions)
- Fanatical Support for AWS environment access: any time a Racker, or one of your employees, accesses your AWS environment by creating a *Passport access request*, you can view the specific instances they had access to, the source of their Passport access request, and other associated details throughout the duration of the Passport access request.

The section of the Fanatical Support for AWS Control Panel provides a timeline-based view of all of these activities. You can view the activities for a specific AWS account or view activities across all of your AWS accounts. You can also filter/facet the results to find the specific activities you are looking for. Logbook retains the last 90 days of historical data indexed for you to explore.

The information in Logbook can prove to be extremely valuable if you need to view the changes your employees, our Rackers, or automated processes made to your environment when troubleshooting an issue or reviewing the root cause of a service impacting event.

CHAPTER

FOURTEEN

CLOUDHEALTH

CloudHealth is the industry-leading cloud management platform that helps you easily visualize cloud spend, quickly view cost savings opportunities and automate cloud governance. Rackspace is the first CloudHealth partner to fully integrate with their platform to include everything from account setup to user management. For example, you can use single sign-on from the Rackspace portal into CloudHealth. Users have full access to CloudHealth with near real-time cost data, savings opportunities, inventory, and much more

CloudHealth is included in Platform Essentials, Navigator and Aviator for all customers. Our Fanatical Support for AWS experts will work with you to:

- Review costs and savings potential
- Suggest tagging strategies to identify departmental owners for spend
- Configure reports for spend breakdown
- Understand cost savings constructs like Reserved Instances and Savings Plans

You can access CloudHealth by clicking the CloudHealth link in the Fanatical Support for AWS Control Panel.

14.1 Permissions

For detailed information on CloudHealth permissions, please see *User Management and Permissions*.

Please contact your Customer Success Manager with any questions related to your CloudHealth and Fanatical Support for AWS Control Panel permissions.

CHAPTER

FIFTEEN

WAYPOINT

Waypoint is a tool that provides high-level, concise information about costs, risks, and other key operational information associated with your AWS accounts. By querying information from various APIs from both AWS and Rackspace, it provides a consolidated view of key information. The goal of Waypoint is to keep you informed about what's happening in your AWS environments and to ensure we work together on improving your experience.

Waypoint is a near-real-time dashboard for the current month, and it also provides historical reports for the previous 12 months, giving you both current and trending information.

For the current month, cost data is updated approximately once per day. Typically, by the tenth day of a month, we finalize the cost data for the previous month (though the process may take a few days longer to run).

Users with access to all AWS accounts within a Rackspace account can access Waypoint by clicking the Waypoint link in the Fanatical Support for AWS Control Panel. Waypoint is available only to users with access to all AWS accounts because it summarizes details for all those accounts.

Users with access to only a subset of all AWS accounts within a Rackspace account can instead access Usage, which provides billing details for the subset of accounts. To access Usage, click the Usage link in the Fanatical Support for AWS Control Panel.

Users will see either Waypoint (for those with access to all AWS accounts) or Usage (for those with access to only some AWS accounts), but not both.

CHAPTER

SIXTEEN

WATCHMAN

AWS CloudWatch is the primary monitoring system used by our Fanatical Support for AWS support teams. CloudWatch provides a wide variety of metrics that cover the entire suite of AWS services - from CPU utilization and disk I/O on EC2 instances to network throughput of your ELB load balancers.

While AWS CloudWatch is available to Fanatical Support for AWS accounts at all service levels, customers using our Aviator service level can opt to have a Racker respond to unexpected deviations in metrics. Watchman is the system responsible for receiving CloudWatch alarms and creating tickets on your Rackspace account.

16.1 CloudWatch Alarms

CloudWatch Alarms can be triggered to fire when the value of a CloudWatch metric deviates from its expected value. For example, if CPU utilization on an EC2 instance exceeds 80% for a period of five minutes or greater, the CloudWatch alarm can be configured to send an alert to a Rackspace-managed SNS (Simple Notification Service) topic (named *rackspace-support*) that will generate a ticket for further investigation by a Racker.

The *rackspace-support* SNS topic is configured in each region when your AWS account is first setup for Fanatical Support for AWS. A subscription to the SNS topic is created for a centralized region-specific SQS (Simple Queue Service) queue that resides in our shared management services account. Our shared management services system continually monitors these queues and generates a ticket when a valid CloudWatch alarm is received from an Aviator service level account.

Note: While the SNS topic described above is present on every Fanatical Support for AWS account, only accounts at the Aviator service level will have tickets generated. If your account is at the Navigator service level, no action will be taken for CloudWatch alarm notifications sent to your account's *rackspace-support* SNS queue.

16.2 Custom CloudWatch Configuration

CloudWatch allows for the creation of custom metrics to allow monitoring the things that are most critical to the uptime of your applications. As an Aviator customer, you can create custom CloudWatch metrics and alarms, as well as send notifications to the *rackspace-support* SNS topic if you desire a Racker response to triggered alarms. We do recommend that you work with a Racker when first creating custom CloudWatch metrics and alarms so that we can ensure that everything is configured properly and that the desired Racker response is clearly documented.

16.3 SmartTickets

SmartTickets is an automation system connected to Watchman that helps provide Rackers additional contextual information in response to CloudWatch Alarms. SmartTickets leverages AWS Systems Manager and AWS APIs to run diagnostic commands against your AWS infrastructure, and displays this information to the Rackers responding to alarms.

SmartTickets activities are recorded in AWS CloudTrail and Rackspace Logbook, and can be identified by examining the API history for the *racker-faws-a63460cb* user.

CHAPTER

SEVENTEEN

AWS INSTANCE SCHEDULER

AWS Instance Scheduler is an AWS-provided solution that enables customers to configure custom start and stop schedules for their EC2 and RDS instances. The solution can help reduce operational costs by stopping resources when they are not needed, and start them back up based on a defined schedule. The most common example is stopping DEV instances outside of working hours (reducing weekly utilization from 168 hours to 50 hours - yielding a 70 percent reduction in running costs).

Instance Scheduler is a "solution". It is not an AWS "service", so you will not find it in the AWS console. It is composed of a number of AWS services that are packaged together into a CloudFormation template. It's important to note that it does not feature a user interface. The solution is developed, maintained and packaged by AWS. It is deployed, configured and managed by Rackspace.

17.1 Use Cases

You might find Instance Scheduler useful if you have any of the following:

- Non-prod environments that are not needed 24/7
- Proof-of-Concept environments that are used for occasional demos
- Nightly batch/job processing environments
- Resources that are started manually by users only when they are needed

17.2 Features

Instance Scheduler supports:

- Stopping and starting stand-alone EC2 instances
- Stopping and starting RDS instances
- Stopping and starting instances in multiple regions
- Starting or stopping instances manually outside the defined schedule
- Partial automation (stop-only or start-only schedule)

It does not support:

- Stopping or starting instances that are part of an auto-scaling group (native ASG scheduled actions can be used in this case)
- Stopping or starting other managed services (e.g. Redshift, Elasticsearch, ElastiCache, etc.)

17.3 Usage

If you would like to start using Instance Scheduler, please submit a request via a Support ticket, or reach out to your Account Manager.

Rackspace will deploy the solution for customers that are interested, and will configure the schedules according to their needs.

Schedule definitions are stored in a DynamoDB table, with every Schedule having a unique name. Resource tags are used to associate EC2 and RDS instances with a particular Schedule (e.g. Schedule=uk-office-hours, when "uk-office-hours" is a name of one of the Schedules we defined in DynamoDB).

If you ever need to "override" a schedule, there are 2 ways to achieve this:

- If it's just for a short period of time (e.g. a few hours), you can simply start/stop the instance via the AWS console. Instance Scheduler will keep applying the defined schedule as before.
- If it's for a longer period of time, you can change the value of the "Schedule" tag key on the instance to either **running** or **stopped**. Before you do that, please take a note of the previous value. Instance Scheduler will keep the instance in this state until you change the value of the tag back to its original value.

17.4 Pricing

Management of Instance Scheduler is provided at no additional cost for Aviator customers. Customers are responsible for the minimal infrastructure cost to run the solution (normally up to $10 per AWS account per month).

CHAPTER EIGHTEEN

SUPPORT

There are multiple ways to receive Fanatical Support for your AWS account(s). A helpful Racker is always just a phone call or ticket away. We are available live 24x7x365.

18.1 Tickets

One of the primary ways that you can interact with a Racker is by creating a ticket in the Fanatical Support for AWS Control Panel. Once logged in, click the Support button in the black bar at the top of the screen and follow the links to create a new ticket or view an existing ticket.

Our automated systems will create tickets for events on your AWS account(s) that require either your attention or the attention of a Racker. For example, our *Rackspace Watchman* will create a ticket when an alarm is raised that requires attention.

Any time a ticket is updated, you will receive an email directing you back to the Control Panel to view the latest comments.

18.2 Phone

Would you prefer to speak to a live Racker? Give our team a call at 877-417-4274 (US) or 0800-033-4045 (UK) and we'll be happy to assist you. Additional international contact numbers are available on our Contact Us page.

CHAPTER NINETEEN

PATCHING

This section describes the necessary considerations and steps for patching guest Operating Systems in AWS services where the OS is unmanaged, with a particular focus on EC2 instances.

Customers of *Fanatical Support for AWS* may wish to reach out to the Rackspace support team for further advice, guidance, and assistance on applying security updates or patches to any of these services.

19.1 Patching Guide for Amazon EC2

The guidelines on this page will assist you in applying guest Operating System updates to Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instances, covering both standalone instances and Auto-Scaling instances in a variety of common deployment models. This document will be updated from time to time, and you may wish to check back regularly - or use this as reference for your monthly patching cycle.

19.1.1 Automation artifacts

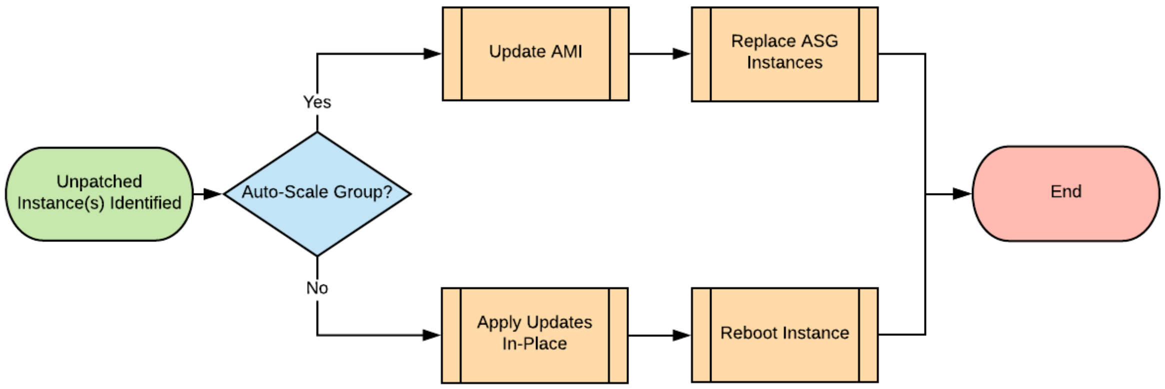
Rackspace recommends using Amazon-published Systems Manager (SSM) documents to manage and apply Operating System updates to standalone EC2 instances. This ensures a repeatable and automatable approach, reducing the possibility of inconsistencies or human errors that might occur when applying patches with direct management access (e.g. SSH/RDP) to instances.

For more information on Amazon-published Systems Manager documents, see Overview of SSM Documents for Patching Instances.

For customers seeking to patch their instances or Auto-Scaling Groups against the January 2018 *Meltdown* and *Spectre* vulnerabilities, Rackspace-developed SSM documents targeting Meltdown/Spectre patching and AMI generation are detailed in *this page*.

19.1.2 Instance categorization

Many organizations have a complex estate in AWS, comprising many different groups of EC2 instances. Guest Operating Systems (OS) running on EC2 can be divided into two categories. The following decision diagram provides an overview of categorizing your EC2 instances, and simplified steps for patching instances in both categories:



1. Standalone instances (not in an Auto Scaling Group)
   - **Method:** Apply OS updates to these instances *in-place*, and reboot them.
   - Standalone instances are normally provisioned where the instance needs to be Stateful, i.e. it is storing configuration or data locally and cannot be replaced. An example would be a build server holding local configuration.
   - You may have heard the term 'Auto-Recovery Instance' used to refer to standalone instances configured with CloudWatch Auto-Recovery alarms, improving resilience/availability. For the purposes of this guide, these instances will be considered to be simple Standalone instances.
2. Instances in an Auto Scaling Group (ASG)
   - **Method:** *Update the Amazon Machine Image (AMI)* that these instances are launched from, and perform a *rolling replacement* of the instances in each ASG.
   - ASG instances are normally provisioned where an instance is Stateless, i.e. all data is stored externally (S3, database) and the instance itself can be replaced at any time. An example would be a group of webservers.
   - ASG instances provide both higher availability than Standalone instances, and the potential to horizontally scale to tens or even hundreds of instances.
   - These instances can and do get replaced with new instances on a regular basis (whether or not you have any Scaling Policies applied to your ASG). Patching existing ASG instances in-place is therefore ineffectual.
   - Don't worry if manual changes have been made to an ASG instance - this guide aims to be as pragmatic as possible, and will cover this (and other) corner cases in many different configurations.

19.1.3 Patching process overview

- Generally speaking, if you have a group of identical webservers they're likely to be provisioned in an ASG. You can examine each ASG (including a list of instances) in the AWS console under:
  - Services –> EC2 –> Auto Scaling –> Auto Scaling Groups
- Individual instances fulfilling utility roles are likely to be provisioned as standalone instances
- If you are unsure in any way - or need to programmatically list/report on your instances - you can examine the tags on each instance. ASG instances have the `aws:autoscaling:groupName` tag key (this is an AWS-reserved tag and cannot be manually modified)
- Example AWS CLI commands:
  - Describe all Auto Scaling instances in us-east-1:

```
aws --region=us-east-1 autoscaling describe-auto-scaling-instances
```

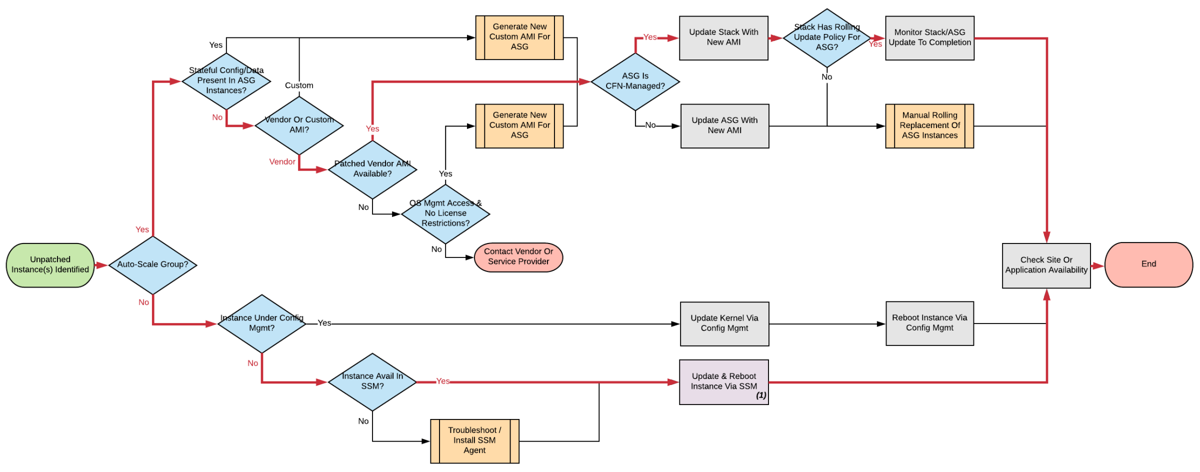
- List all Auto Scaling instances in us-east-1, along with their ASG name and CloudFormation stack name:

```
aws --region=us-east-1 ec2 describe-instances \
    --filters 'Name=tag-key,Values=aws:autoscaling:groupName' \
    --query 'Reservations[*].Instances[*].[InstanceId,Tags[?
↪Key==`aws:autoscaling:groupName`],Tags[?Key==`aws:cloudformation:stack-
↪name`]]'
```

- List all standalone instances in us-east-1, along with their CloudFormation stack name:

```
aws --region=us-east-1 ec2 describe-instances \
    --query 'Reservations[*].Instances[?!not_null(Tags[?Key ==␣
↪`aws:autoscaling:groupName`])].[InstanceId,Tags[?
↪Key==`aws:cloudformation:stack-name`]] | []'
```

Once you have identified whether you are dealing with a Standalone instance or an Auto Scaling Group of one or more instances, you're ready to follow the remediation process below. Don't worry if this looks intimidating at first - the majority of instances will follow one of two simplified paths (one for Standalone, one for ASG instances) which have been highlighted in red. Processes that are automatable with the SSM documents are shaded in purple:


Unpatched Instance(s) Identified
Auto-Scale Group?
Yes
No
Stateful Config/Data Present In ASG Instances?
Yes
No
Custom
Vendor Or Custom AMI?
Vendor
Patched Vendor AMI Available?
Yes
No
OS Mgmt Access & No License Restrictions?
Yes
No
Contact Vendor Or Service Provider
Generate New Custom AMI For ASG
Generate New Custom AMI For ASG
ASG Is CFN-Managed?
Yes
No
Update Stack With New AMI
Update ASG With New AMI
Stack Has Rolling Update Policy For ASG?
Yes
No
Monitor Stack/ASG Update To Completion
Manual Rolling Replacement Of ASG Instances
Instance Under Config Mgmt?
Yes
No
Update Kernel Via Config Mgmt
Reboot Instance Via Config Mgmt
Instance Avail In SSM?
Yes
No
Update & Reboot Instance Via SSM
(1)
Troubleshoot / Install SSM Agent
Check Site Or Application Availability
End

(Click through for a larger version of this or any other image)

19.1.4 Standalone instances

Standalone instances should be patched in-place and rebooted.

If the instance is managed by Configuration Management (examples: Puppet, Chef, Ansible, Saltstack), you will likely want to simply use this configuration management to apply updates and reboot. Good news: You're done! Time to move on to the next instance.

If the instance is not under Configuration Management, you should apply the updates using Amazon Systems Manager (also known as Simple Systems Manager or SSM), or manually using native OS tools if necessary.

Apply OS patches

Systems Manager Patch Manager walkthroughs can be found in the AWS Systems Manager User Guide.

If necessary, the updates can be manually checked, applied, instance rebooted and updates validated through your usual management access to this instance (SSH/RDP). However, Rackspace recommends using the Systems Manager Documents to ensure repeatability, eliminate manual work and manage scheduling across your EC2 instance estate.

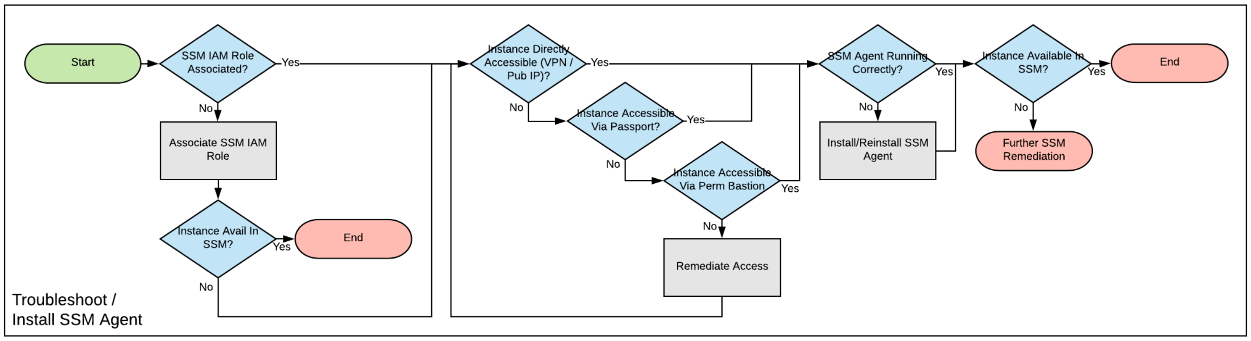
Amazon Systems Manager Troubleshooting

If your instance is not available in Systems Manager, this is likely caused by one of the following two problems:

#. The instance IAM role does not allow the instance to communicate with the Systems Manager API

- The instance role or role policies can be updated using the AWS console, API or CLI without any downtime to the instance
- Customers of *Fanatical Support for AWS* who consume our *Aviator* service offering should find the *RackspaceDefaultEC2Role* and *RackspaceDefaultEC2Policy* created on all accounts provide the correct permissions
- If for any reason you need to create a role and/or policy manually, please see Create an Instance Profile Role for Systems Manager in the Amazon Systems Manager User Guide, or reach out to the *Fanatical Support for AWS* Support team for assistance

1. The SSM Agent is not running on the instance
   - This will necessitate accessing the instance (usually via SSH or RDP) and reinstalling the agent
   - Documentation on installing the SSM agent can be found in the AWS Systems Manager User Guide


Start
SSM IAM Role Associated?
Yes
No
Associate SSM IAM Role
Instance Avail In SSM?
Yes
End
No
Troubleshoot / Install SSM Agent
Instance Directly Accessible (VPN / Pub IP)?
Yes
No
Instance Accessible Via Passport?
Yes
No
Instance Accessible Via Perm Bastion
Yes
No
Remediate Access
SSM Agent Running Correctly?
Yes
No
Install/Reinstall SSM Agent
Instance Available in SSM?
Yes
End
No
Further SSM Remediation

If you need to troubleshoot further, or manually resolve either issue, please see Where Are My Instances? in the Amazon Systems Manager User Guide.

19.1.5 Auto Scaling Group instances

Auto Scaling Groups should be updated with a new Launch Configuration (LC) specifying a new AMI (incorporating the necessary OS updates), and replaced via a rolling update to the ASG.

Update AMI

Auto Scaling Groups under CloudFormation management should be updated with a CloudFormation Stack Update.

- CloudFormation management can be confirmed by checking the Auto Scaling Group tags for the `aws:cloudformation:stack-name` tag key (this is an AWS-reserved tag and cannot be manually modified)
- Example AWS CLI commands:
  - List all ASGs in us-east-1 created by CloudFormation, and the CloudFormation stack name:

```
aws --region=us-east-1 autoscaling describe-auto-scaling-groups \
    --query 'AutoScalingGroups[?not_null(Tags[?Key ==␣
↪`aws:cloudformation:stack-name`])].[AutoScalingGroupName,Tags[?
↪Key==`aws:cloudformation:stack-name`].Value] | []'
```

  - List all ASGs in us-east-1 not created by CloudFormation:

```
aws --region=us-east-1 autoscaling describe-auto-scaling-groups \
    --query 'AutoScalingGroups[?!not_null(Tags[?Key ==␣
↪`aws:cloudformation:stack-name`])].AutoScalingGroupName | []'
```

- Almost all CloudFormation stacks use a template that allows entering the AMI ID as a parameter during the stack update
  - AWS Console –> Services –> CloudFormation –> StackName –> Update Stack
  - Advance to the Specify Details stage and look for an 'AMI' or 'Image ID' parameter
- A very few CloudFormation stacks may hard-code the `ImageId` property of the `AWS::AutoScaling::LaunchConfiguration` resource (which is in turn referenced by the `AWS::AutoScaling::AutoScalingGroup` resource)
  - These stacks should be updated by changing the `ImageId` property in the template and updating the stack with the new template
  - If you're having trouble updating a CloudFormation template - or need to move towards best practices, like parameterizing the AMI ID, then Rackspace may be able to assist. Please reach out to us!
  - If a change has been made to a template, remember to check this into any version control repository in use (e.g. git)

Auto Scaling Groups not under CloudFormation (or other Infrastructure as Code) management should be updated by creating a new Launch Configuration (LC) and manually applying this to the Auto Scaling Group:

1. Identify the current LC used for the ASG
2. Create a copy of the LC, with an updated AMI #. AWS Console –> Services –> EC2 –> Auto Scaling –> Launch Configurations #. Select a LC –> 'Copy launch configuration' #. 'Edit AMI' #. 'Create launch configuration'
3. Edit the ASG and select the new LC

Vendor AMI

If you are using a default vendor AMI with no 'baked in' customization, then you can simply update the ASG with the latest version of the vendor AMI. The documentation/lists are linked for convenience below, but for the avoidance of doubt, the latest AMI issued by the vendor should always be used.

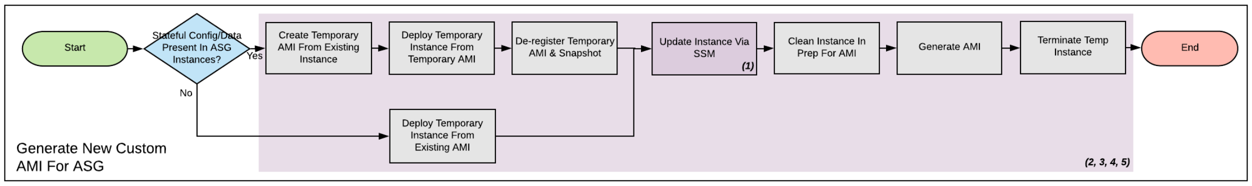
- Amazon Linux - AMI list
- Red Hat Enterprise Linux - How to list AMIs
- The Ubuntu operating system - AMI list
- CentOS Linux - How to list AMIs
- Microsoft Windows - AMI list

Generating a custom AMI

Under the following circumstances, it may be necessary to generate a custom AMI for your ASG:

1. The ASG is already using your own custom AMI (usually a descendant of an original vendor AMI)
2. No updated vendor AMI is available
3. Manual changes have been made to the ASG instances (usually using direct SSH/RDP access), and these changes have not been integrated into the existing AMI, User Data, CloudFormation template or Configuration Management
    - Manual changes such as these would be lost when the instances are replaced
    - **This is a precarious situation, since the instances might get replaced at any time**, even outside of this patching process (e.g. An existing instance goes down, a routine scaling event, etc)
    - It is therefore critically important to integrate these changes by creating a new custom AMI

The below process outlines generating a custom AMI (if necessary). As before, automatable processes are shaded in purple - SSM documents can be used to automate the process of generating a new patching AMI, either from an existing AMI or from an existing instance. For examples, please see the *Rackspace SSM documents* targeting AMI generation for Meltdown/Spectre remediation.


Start
Stateful Config/Data Present In ASG Instances?
Yes
No
Create Temporary AMI From Existing Instance
Deploy Temporary Instance From Temporary AMI
De-register Temporary AMI & Snapshot
Update Instance Via SSM
(1)
Clean Instance In Prep For AMI
Generate AMI
Terminate Temp Instance
End
Deploy Temporary Instance From Existing AMI
Generate New Custom AMI For ASG
(2, 3, 4, 5)

If for an reason you cannot use the Rackspace-authored SSM documents, below is a walk-through of the manual steps needed:

1. If manual changes have been made to ASG instances, it is necessary to make a 'temporary' AMI from one of the existing instances in order to capture these manual changes
    - You may wish to do this offline (i.e. Using the `--reboot` CLI argument, or without choosing `No reboot` in the console AMI generator wizard) to ensure the instance is shut down properly for a consistent filesystem snapshot
2. Deploy a temporary instance from either your current AMI, your vendor AMI, or the temporary AMI generated in step 1
    - If you generated a temporary AMI in step 1, you can deregister it and remove the associated EBS snapshot now
3. Patch this temporary instance as any standalone instance
    - If the temporary instance is available in SSM, use the instructions under Apply OS patches above to update the instance using SSM documents
    - If the temporary instance is not available in SSM (e.g. The AMI did not contain an installation of the SSM agent), it will be necessary to access the instance directly (SSH/RDP) and manually apply updates
4. Prepare the temporary instance for AMI generation by removing data, configuration and software that will be deployed by your ASG instance launch and bootstrapping code
    - Examples of items you may need to remove:
        - SSH keys and other secrets
        - Log files
        - Application code
        - Software agents

5. Generate an AMI from this temporary instance using the AWS console, AWS CLI, or any third-party tool that can call the `CreateImage` API function
6. Terminate the temporary instance

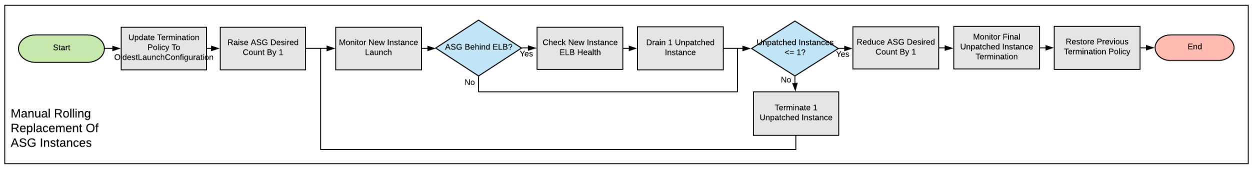
Rolling replacement of instances

Most ASGs will be updated using CloudFormation, and the stack template will contain a RollingUpdate `UpdatePolicy` for the ASG. If this is the case, CloudFormation will manage the rolling replacement of your instances - deploying a new instance, waiting for it to pass ASG Health Checks, draining and terminating an old instance. You will see a message similar to the following in the stack *Events* following the stack update, and can monitor the stack update to completion:

```
Rolling update initiated. Terminating 5 obsolete instance(s) in batches of
1, while keeping at least 4 instance(s) in service. Waiting on resource
signals with a timeout of PT20M when new instances are added to the
autoscaling group.
```

Manual rolling replacement

If the ASG is not managed by CloudFormation - or the CloudFormation stack template does not contain a RollingUpdate `UpdatePolicy` for the ASG - then you will need to perform a manual rolling replacement of the instances in the ASG. This process is illustrated using the diagram below:


Start
Update Termination Policy To OldestLaunchConfiguration
Raise ASG Desired Count By 1
Monitor New Instance Launch
ASG Behind ELB?
Yes
No
Check New Instance ELB Health
Drain 1 Unpatched Instance
Unpatched Instances <= 1?
Yes
No
Terminate 1 Unpatched Instance
Reduce ASG Desired Count By 1
Monitor Final Unpatched Instance Termination
Restore Previous Termination Policy
End
Manual Rolling Replacement Of ASG Instances

Alternatively, you may wish to update the CloudFormation stack template to add an `UpdatePolicy` to the Auto Scaling Group resource, similar to the following:

```
"UpdatePolicy": {
  "AutoScalingRollingUpdate": {
    "PauseTime": "20M",
    "WaitOnResourceSignals": "true",
    "SuspendProcesses": [
      "ScheduledActions",
      "ReplaceUnhealthy",
      "AlarmNotification",
      "AZRebalance"
    ],
    "MaxBatchSize": "1",
    "MinInstancesInService": "1"
  }
}
```

More information and examples can be found in the CloudFormation User Guide: AutoScalingGroup and UpdatePolicy.

In-place patching of ASG instances (emergency only)

EC2 instances running under an ASG *can* be patched/rebooted in place using the SSM documents above, but it is important to consider this as **emergency mitigation only**. This must be followed by an AMI update and rolling replacement of all instances as soon as possible. New instances launched at any time in the future will be unpatched (*even if automatic patching is enabled*), and there will be a configuration discrepancy with any existing instances, which will be in an untested configuration.

More information: Correct functioning of your application within a group of Auto-Scaling instances relies upon current running instances and instances launched at any future date/time holding the same configuration. As this configuration is made through in several stages or layers (examples below), synchronizing and adequately testing existing instances against the configuration for future instances can be very difficult and error-prone. Rackspace's recommended best practice is therefore to update the underlying AMI and perform a rolling replacement of all Auto-Scaling instances, as described in this guide.

Instance launch configuration stages:

1. AMI: An instance is launched from the AMI in the Launch Configuration
    - Commonly a plain vendor AMI or a customized 'silver'/'gold' AMI pre-configured with some software packages or application code
2. Cloud-init: The EC2 service uses cloud-init to perform initial instance configuration
    - Includes resetting OS configuration left in the AMI, setting up networking, deploying SSH keys, etc
3. User Data: cloud-init then executes the User Data (often used to setup software repositories, configuration management agents, etc)
4. Bootstrapping: Installation and configuration of software packages etc, usually using CloudFormation cfn-init metadata and/or your Configuration Management
5. Application Deployment: Copying and testing your application code, using AWS CodeDeploy, Configuration Management, or another dedicated deployment agent

19.2 Automation Artifacts for Patching Meltdown/Spectre

Rackspace has developed several Amazon Systems Manager documents to help automate patching and AMI generation tasks. For customers seeking to patch their instances or Auto-Scaling Groups against the January 2018 *Meltdown* and *Spectre* vulnerabilities, particularly useful documents are listed below. These can be leveraged as part of the process described in the *Patching Guide for Amazon EC2* to patch vulnerable instances in place, or to generate patched AMIs from existing instances or AMIs.

Customers of *Fanatical Support for AWS* who consume our *Aviator* service offering can find these available as Systems Manager Shared Resources (in the AWS console, browse to EC2 –> Systems Manager Shared Resources –> Documents –> Private Documents and look for documents labeled with the owner '507897595701').

Customers of *Fanatical Support for AWS* who consume our *Navigator* service offering can download these from the direct links below.

1. FAWS-MeltdownSpectre-PatchRunningWinLinuxEC2
    - *Checks for presence of meltdown/spectre patch for Windows and Linux machines. Optionally apply patches also.*
2. FAWS-MeltdownSpectre-PatchLinuxAMI
    - *Creates new patched AMI (Spectre/Meltdown) from Linux source AMI*
3. FAWS-MeltdownSpectre-PatchWindowsAMI

   - *Creates new patched AMI (Spectre/Meltdown) from Windows source AMI*
4. FAWS-MeltdownSpectre-PatchLinuxEC2toAMI
   - *Creates new patched AMI (Spectre/Meltdown) from running EC2 instance*
5. FAWS-MeltdownSpectre-PatchWindowsEC2toAMI
   - *Creates new patched AMI (Spectre/Meltdown) from running Windows EC2 instance*

19.3 Patching Amazon ECS

Rackspace's recommended method for securing ECS clusters with current software updates is to update the environment to the latest available Amazon ECS-Optimized AMI. Following this, rotate (replace) the host instances following the methods described in the *Patching Guide for Amazon EC2* document.

Note: If an ECS cluster is using a custom AMI, this AMI will need to be rebuilt with the necessary updates prior to updating the environment with the new AMI and following the rest of the recommended method above.

19.4 Patching AWS Batch

19.4.1 Managed Compute Environments

Rackspace's recommended method for securing AWS Batch *managed compute environments* with current software updates is to create a new compute environment (defaults to the latest Amazon ECS-Optimized AMI, or rebuild a new custom AMI). This new compute environment can then be added to an existing AWS Batch job queue, and the old compute environment removed and deleted.

19.4.2 Unmanaged Compute Environments

For *unmanaged compute environments*, you manage your own compute resources as an ECS cluster, and should follow the *Patching Amazon ECS* guidelines to update the ECS cluster.

For more information, see Compute Environments in the AWS Batch User Guide.

19.5 Patching Amazon EMR

19.5.1 Recommended Method

Rackspace's recommended method for securing EMR clusters with current software updates is to launch a new EMR cluster, using the latest EMR release. EMR clusters are not intended for indefinite runtime and are expected to terminate after a specific processing job (*Transient clusters*) or after a succession of jobs (*Long-running clusters*).

Note: If an EMR cluster is using a custom AMI, a new AMI should be create including the desired updates, and a new EMR cluster launched with the new AMI (the AMI used for an EMR cluster cannot be altered after launch).

19.5.2 Alternative Method

It is possible to manually patch cluster instances in-place using OS management access to the instances. This may be appropriate if an EMR cluster needs to remain provisioned for a recurring job, but security updates are critical. The EMR cluster should be replaced using the recommended method above as soon as a downtime window is available.

Note: Any new instances that are launched - either through scaling changes or self-healing - will apply outstanding patches at launch.

19.6 Patching AWS Elastic Beanstalk

Rackspace's recommended method for securing Elastic Beanstalk environments with current software updates is to update the environment in-place to the latest version of the environment's platform. Elastic Beanstalk will update the Auto-Scaling Group and Launch Configuration with the new AMI version and will perform a rolling replacement of instances. For more information, see Updating Your Elastic Beanstalk Environment's Platform Version and Elastic Beanstalk Supported Platforms.

19.6.1 Managed Platform Updates

Additionally, you may wish to enable managed platform updates, which can update the environment's platform version inside a weekly maintenance window (not available for the .NET on Windows Server platform).

19.7 Patching AWS OpsWorks Stacks

19.7.1 Recommended Method

Rackspace's recommended method for securing OpsWorks Stacks with current software updates is to replace the instances with new instances. New instances will have the latest set of security patches installed during bootstrapping. Once the new instances are online, the old instances can be deleted. For more detailed information on this or the alternative options listed below, see Managing Linux Security Updates in the AWS OpsWorks User Guide.

19.7.2 Alternative Methods

1. Run the Update Dependencies stack command (Chef 11.10 or older stacks), which will install updates in-place on the instances.
2. Set the InstallUpdatesOnBoot parameter to false for the instances or layer, and install updates manually using OS management access.

CHAPTER

TWENTY

AWS MARKETPLACE

You are able to purchase items from the AWS Marketplace for use in your AWS account. Note that any purchases from the AWS Marketplace will be billed to you along with your AWS infrastructure and Rackspace management fees on your monthly invoice. Additionally, any purchases from the AWS Marketplace will calculated as AWS infrastructure for purposes of calculating Rackspace management fees.

20.1 Legal Terms

We may agree to install third party software (for example, from an AWS marketplace) as part of the Services. Where such activity requires the acceptance of an End User License Agreement (or similar terms), you hereby authorize Rackspace to accept such terms on your behalf, agree to be bound by and adhere to such terms, and acknowledge that you, and not Rackspace are bound by such terms. We will notify you via ticket when we accept such terms on your behalf and direct you to a copy.

CHAPTER TWENTYONE

CLOUD NATIVE SECURITY

AWS is constantly expanding its portfolio of native security products. Thanks to its more holistic access to backend data and resources, Amazon is able to offer security features that 3rd party tools are unable to provide. Rackspace is expanding its managed security services to provide support for native AWS security products.

AWS customers who want to improve their security posture by using products like AWS Security Hub, Amazon GuardDuty and IAM Access Analyzer but do not have the expertise or the resources to invest in a 24x7x365 Security Operations Center (SOC) are now able to utilize the Cloud Native Security Service Block from Rackspace.

Amazon GuardDuty provides threat detection utilizing existing AWS native data sources (CloudTrail Logs, VPC Flow Logs, and DNS Logs) without any potentially disruptive deployment steps such as agent installation. AWS describes GuardDuty as a service that "uses machine learning, anomaly detection, and integrated threat intelligence to identify and prioritize potential threats".

IAM Access Analyzer allows customers to verify that their policies provide exactly the level of access they need. It continuously monitors for new or updated policies, and analyzes permissions granted using policies for Amazon S3 buckets, AWS KMS keys, Amazon SQS queues, AWS IAM roles, and AWS Lambda functions.

AWS Security Hub provides the aggregation point for GuardDuty and IAM Access Analyzer findings across multiple AWS accounts, acts as the conduit to Rackspace systems, and serves as a single pane of glass for all native security services which emit findings. It also provides compliance standard checks against industry best practices (e.g. CIS AWS Foundations). Security Hub will be enabled for all Cloud Native Security customers, but the compliance standards feature will be optional.

This Service Block requires Security Hub, but the other native security products are optional and can be disabled by creating a ticket with Rackspace.

21.1 Onboarding

During onboarding to this Service Block, Rackspace will deploy and configure the security products onto customer's AWS account(s) across different regions. Rackspace will also collaborate with the customer to create a customer-specific runbook.

Every customer will have a dedicated AWS account to be used as Security Hub master and GuardDuty master. Rackspace will enable and configure Security Hub and GuardDuty on the master account and all other accounts that are in scope, including configuration of the master-member relationship.

Rackspace will configure Security Hub to ingest security findings from GuardDuty and IAM Access Analyzer. It will also be configured to deliver the events to the Rackspace SIEM (Security Information and Event Management), a system that is used by our security specialists to monitor the security of customer environments 24x7x365.

A runbook will be created for every customer. The purpose of the runbook is to provide Rackspace engineers with information about the customer's environment and allow them to apply context to security findings. A typical runbook might include:

- Information about the environments (what is prod/non-prod, which environments contain sensitive data, any expected "suspicious" traffic to remote regions or crypto-currency activities)
- Customer escalation process (who to contact and when)
- Any actions Rackspace can perform as part of remediation without explicit approval (e.g. isolate compromised instances, invalidate compromised credentials, patch vulnerabilities)
- Any known IP whitelists or threat lists

21.2 Monitoring

Rackspace security specialists will monitor Security Hub, GuardDuty and IAM Access Analyzer findings 24x7x365. Rackspace will begin analysis of findings within response times commensurate with the finding severity:

- Critical: 30 minutes
- High: 60 minutes
- Medium: 4 hours

Rackspace security specialists will analyze and filter the findings and will apply context based on customer's runbook – for example:

- Identification of false-positive findings
- Identification of affected environment – production/test/dev
- Aggregation of multiple related findings into a single incident
- Review of the findings against customer's runbook

Once the findings have been analyzed, escalation and notification processes will be initiated.

21.3 Investigation and Remediation

Rackspace engineers will investigate the findings, engage the customer as necessary according to the runbook, and recommend remediation actions. On AWS accounts that include the Manage & Operate Service Block, Rackspace will remediate the findings.

21.4 Ongoing Management

When customers create new AWS accounts, they will need to request Rackspace via a ticket to enable and configure the security products on these newly created AWS accounts.

Rackspace will provide ongoing management of the security products in a customer's environment. This might include activities such as creation of custom Security Hub Insights based on customer's security needs and management of GuardDuty auto-archive rules and trusted IP/threat lists according to the customer's requirements.

In order to ensure that the coverage of the service remains sufficient for the customer's requirements, Rackspace will perform monthly account reviews. This will allow customers to adjust the level of service and the mix of the security products as their requirements change. Account reviews might include: review of the customer's runbook, summary of findings and remediations, compliance status, review of reoccurring findings, review of IP lists.

21.5 Minimum Severity Level

Every security finding processed by Security Hub includes an associated normalized severity level – Critical, High, Medium, Low, Informational.

In order to give customers some control over costs (which have a direct correlation with the amount of findings Rackspace responds to) and to help with prioritization, we will define a minimum severity level for each customer. Rackspace will only respond to findings that are at or above the defined minimum severity level. Findings with lower severity level will not be processed by Rackspace and the customer won't be charged for those. The minimum severity level will be agreed upon between the customer and Rackspace.

Initial selection of the minimum severity level for Rackspace response can either be based on a review of the native security products in the customer's environment (if they are already enabled) or simply be set to the highest severity level (i.e. Critical/High) to start with. As part of the regular monthly account reviews, the minimum severity level will be reviewed by Rackspace and the customer. When the current setting does not generate too many findings, we can reduce the minimum severity level and start responding to findings with a lower severity. Likewise, if the current setting is generating too many findings, we can increase the minimum severity level.

21.6 Billing

Cloud Native Security has a utility billing model. The monthly charge for the service is calculated as the product of the number of events multiplied by the cost per event, or the minimum monthly fee, whichever is greater.

Native security products produce findings. Findings that are at or above the defined minimum severity level are sent to Rackspace SIEM, where they go through correlation rules and transformed into events. Every event includes one or more correlated findings. Rackspace security specialists will respond to these events. The per-event fee will apply to events (SIEM-correlated findings). For example, if the native security products generate five findings that are delivered to the SIEM and, after correlation, the SIEM generates two events (one for two related findings and one for three related findings), the customer will be charged for two events only.

Cloud Native Security service fees do not include the cost of the native security products themselves. The cost of the native products as well as the costs of supporting infrastructure components that are required to integrate the native products with the Rackspace SIEM are included in the AWS infrastructure charges portion of the customer's Rackspace bill.

Initial deployment and configuration will be charged as a one-time onboarding fee.

21.7 Additional Services

Cloud Native Security (as the name suggests) provides managed security services using native AWS security products. Customers interested in improving their security posture over and above what is possible from the Cloud Native Security Service Block alone can purchase the Rackspace Proactive Detection and Response service block. Proactive Detection and Response includes all the capabilities of Cloud Native Security, plus additional 3rd party products, agent-based OS level threat detection, threat hunting and threat intelligence.

CHAPTER

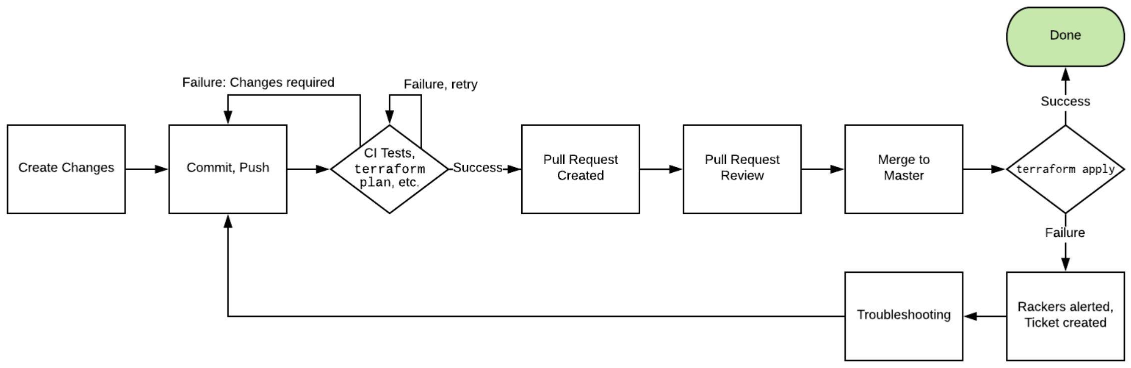
TWENTYTWO

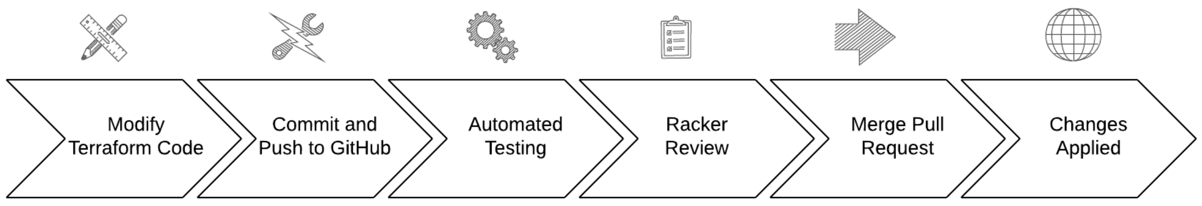
MANAGED INFRASTRUCTURE AS CODE

Deployment through infrastructure-as-code (IaC) helps you build and manage cloud infrastructure through code, utilizing software development best practices and industry-standard DevOps tools and techniques. Using an industry-standard tool called Terraform, your infrastructure code is peer reviewed through pull requests, and committed to version control, all while being tested and deployed through automation steps called continuous integration (CI) and continuous delivery (CD).

This means that changes to your environment are managed through this specialized Terraform workflow, and not through the AWS Console directly. Changes in the AWS Console can conflict with Terraform management, resulting in downtime, data loss, or delays to reconcile these manual changes. It is important that all changes to your environment are managed with Terraform. If you need assistance applying a change, the Rackspace Support team will be happy to help.

22.1 Change Workflow





22.1.1 Initiating a change

You may begin the change process via a new pull request or by creating ticket; if you choose to make a pull request, Rackers will be alerted to your pull request and may comment or make additional comments within the pull request. Even if you request a change by creating a ticket, a pull request will be created with your changes, which you may review. You must have a GitHub account if you would like to participate in pull requests.

After you've made your Terraform changes, committed them using git, and pushed them up to GitHub.com, your repository will automatically begin building the changes. Specifically, changes will be checked against style rules (also known as linting), and a `terraform plan` will be run for each layer that contains changes. You may create a pull request at any time while these steps are running.

Pull requests are at the heart of making a change in an infrastructure as code world. A GitHub concept, pull requests, "is how you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch." Detailed review and conversation about changes is encouraged in GitHub.

You can also open a draft pull request for work that is in Progress. A draft PR (pull request) will not create a notification to Rackspace. Once you are ready to merge, mark the pull request as ready for review and it will generate a notification for a Racker to review your changes.

There are two required checks that must pass for your pull request to be merged:

1. Approval (`rackspace/approval`): The pull request must be approved by at least one Racker, using the GitHub PR review mechanism. Approvals by the committer will not count towards this requirement.
2. Plan (`ci/circleci: plan`): The branch must successfully pass `terraform fmt` and successfully run *terraform plan*. If either of these steps fail, you may push additional commits to address them.

22.1.2 Approving a change

Rackers must approve pull requests, before any changes are merged to the `master` branch. If you would prefer to approve any changes or pull requests as well, please explicitly state this in a ticket or pull request. If you would also like to have a change deployed on a specific schedule, please note that, too.

When Rackers review a pull request, they are looking to understand:

- Did all automated tests and checks pass? Are the code changes technically correct?
- Does this change do what was requested, in an appropriate and well-designed way?
- Can Rackspace proceed with making the change (now or at a specific time)?
- Does this change require any additional, explicit approval from the requestor?

22.1.3 Deploying a change

Changes are deployed when a pull request is merged by you or by Rackspace (adding new commits to the `master` branch). Using the layers concept described in the Terraform standards in this site, each layer's `terraform plan` is repeated, and then `terraform apply` is run to finish that layer. If Terraform encounters any problems during this process, the entire build of the `master` branch will be marked failed, and an emergency Rackspace ticket will be created for Rackers to investigate.

Additionally, please consider the following when deploying a change or performing actions on a pull request:

- Any approvals will be dismissed if additional commits are added to the scope of the pull request.
- Only one pull request will be merged to master, and changes applied, at any given time. Attempts to do more than one may fail.

- Any changes that may be staged far in advance may require re-planning or branch updates at the time of the change. Please be sure to inform Rackspace, in a Support ticket, to schedule for a specific change window or maintenance window. Rackspace may require additional time to update an older pull request as part of a scheduled change.
- Rackspace will not automatically deploy changes without Racker interaction or Racker approval, even if two reviewers approve the pull request. Rackspace doesn't have a default list of changes that can bypass Racker approval.

To raise issues, questions, and changes that aren't already represented as pull requests, Customers should open a new ticket with Rackspace Support. GitHub's Issues feature is disabled on all Rackspace-managed repositories.

22.2 Using GitHub

Rackspace-wide projects containing documentation and/or reusable code to build your infrastructure are public and available in the rackspace-infrastructure-automation organization on GitHub.com. These repositories generally start with a `rackspace-` or `aws-` prefix to let you know that they are public, shared, and maintained by Rackspace.

If you'd like to have access to the repositories that house your infrastructure code, please make a request with your Support team and provide your GitHub.com username and the level of access you would like to have (read, or read-write). Note that all members of the rackspace-infrastructure-automation organization must have two-factor authentication enabled for their GitHub.com account. Once your Support team receives the request, they will create an invitation that you will need to accept to finish the process of getting access. You may accept this invitation through a link sent to you by email, or by simply going to the repository URL provided by our internal team.

In the event that a user's GitHub.com access needs to be revoked – for example, when an employee leaves the company – please also make a request with your Support team, and they can remove this user from the organization.

22.2.1 My Repositories

Once you have access through the process above, you'll find two main types of repositories in GitHub.com under the rackspace-infrastructure-automation organization, in addition to the reusable code mentioned above:

1. *Shared Repository*: These are repositories that are shared across all of your accounts in AWS. These are typically reusable Terraform modules that can be applied to more than one AWS account. These start with your customer number, mention the specific cloud provider, and contain a human readable ending. They are unique to you as a customer, and can't be seen by other customers.

   Unless you have managed services across multiple clouds with Rackspace, you will normally only have one 'shared' repository.

   Example: `12345-aws-LargeCorp`

2. *Account Repository*: These are repositories that map directly to an AWS account. They house Terraform files that are used to directly test and deploy infrastructure in that specific account. These start with your customer number, mention the specific AWS account, and contain a human readable ending. They map directly to the Aviator, Infrastructure as Code accounts we manage for you.

   Examples: `12345-aws-9876541-Production1`, `12345-aws-9876541-Test1`, `12345-aws-9876541-Dev1`

22.2.2 Branches, forks, and pull requests

Rackspace employs continuous integration and delivery (CI/CD) to deploy your infrastructure based on the `master` branch of your repository. Therefore, the `master` branch should always reflect a consistent, deployable, and current expected state of your infrastructure. The `master` branch is also protected from unreviewed changes, and should only be modified through a pull request process.

Other branches should be short-lived, and used to propose changes using a pull request into `master`. Branches from forks will not be built (tested or deployed), and we recommend you simply push to a branch inside the main repository. When the shared repository contains code or files needed by an account repository, Rackspace employs tags using the "GitHub release" mechanism.

22.2.3 Repository layout

Here is a graphical representation of what you can expect to find in your repository. Not all directories and files may be present in all types of repositories. Below, you'll find an explanation of what each item is.

```
- README.md
- .circleci/config.yml
- .terraform-version
- docs
- layers/_main/main.tf
- modules/example/main.tf
```

- **README.md** and **docs/**: All repositories are created with a default readme file and documentation directory. This is used for any documentation or reference material that you wish. Rackers working on your requests will be able to refer to this material as needed.
- **.circleci/config.yml**: This file configures the CI/CD system with specific workflow steps to execute, and a defines a container in which to run them. This file should not be modified except through coordination with Rackspace.
- **.terraform-version**: This file is used by our tooling and automation to call Terraform Version Manager (tfenv), which ensures only a specific, intended version of the Terraform CLI will be used when building and deploying your infrastructure.
- **layers/**: This directory will be present on *Account Repositories*, and contains different groupings of Terraform files that make up one 'state'. When your infrastructure is deployed, only layers that contain changes will be tested and deployed. A more detailed explanation of this design can be found on the Terraform section of this guide.
- **modules/**: This directory will be present on your *Shared Repository* and contains Terraform modules intended to be reused.

Note that if you run Terraform locally, you may also see a **.terraform** directory. This contains configuration and data that should not be committed to the repository. This ensures our build process always fetches the latest providers, modules, and state configuration needed to build your environment.

22.3 Using Terraform

22.3.1 What is Terraform?

> "Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions." – [Introduction to Terraform](https://www.terraform.io/intro/index.html)

22.3.2 Making changes with Terraform

Rackspace strongly recommends that all changes be made through CI/CD tooling, and Terraform should not be run locally except in extreme cases, especially `terraform apply`. Because all repositories have a **.terraform-version** file, and are named appropriately for a specific AWS account, our tooling will ensure that the correct version of Terraform is executed against the correct account.

As mentioned in the *Using GitHub* section of this documentation, there is also a shared repository for Terraform modules you may wish to reuse across multiple accounts. Rackspace will create "GitHub release" objects in this repository, which automatically makes tags that we can use to reference specific modules at specific points in time.

Please see the later part of this document for specific standards Rackspace recommends when creating Terraform modules and Terraform configuration.

22.3.3 Where is Terraform state stored?

Rackspace maintains a separate S3 bucket for storing the Terraform state of each AWS account. Access to this bucket and its contents is restricted to the `Rackspace` role that Rackspace maintains on every AWS account. By default, temporary credentials retrieved from your control panel will use the Rackspace role and automatically have access to this bucket, should you need it. You can read more about the Rackspace role in the *AWS account defaults* section of this documentation.

In addition, Rackspace stores these buckets in an entirely isolated AWS account, and implements best practices such as requiring bucket encryption, access logging, versioning, and preventing accidental public access. Because S3 bucket names are globally unique, and as part of a defense-in-depth strategy, we choose an arbitrary, opaque name for this bucket that cannot be mapped back to an AWS account. We provide the bucket name in the logged output from each CI/CD job, as well as the full terraform commands we run, should you want to inspect it or use it to run Terraform locally.

22.3.4 Grouping state into layers

There are a few different designs employed in the Terraform community for how to structure your Terraform files, modules, and directories. The community around Terraform has written blog posts and spoken at HashiConf about how these patterns evolve over time; many of the recommendations focus on how to best group Terraform state. At Rackspace, we've built upon the existing best practices (e.g. 'module-per-environment') and created a concept we call *layers* in order to isolate Terraform state.

What is a layer?

Put simply, a layer is a directory that is treated as a single Terraform configuration. It is a logical grouping of related resources that should be managed together by Terraform. Layers are placed in the **layers/** directory inside an *Account Repository*. Our automation will perform all of the usual Terraform workflow steps (init, plan, apply) on each layer, alphabetically.

In collaboration with experienced Rackers, you should carefully consider how to logically group the state of your AWS resources into layers; layers could represent environments like production or test, regions your application may be hosted in, application tiers like "database" or "web servers," or even applications that share availability requirements.

Here are some considerations that Rackers will discuss with you when planning out your environment:

- ensure resources frequently modified together are also grouped together in the same layer
- keep layers small, to limit blast radius and ensure refreshing state is quick/safe
- keep dependencies between layers simple, as changes must take dependencies into consideration manually

- consider reading state from another layer, using a data source; never write to another layer's state
- for small environments, consider that a single layer may acceptable, but moving resources between layers is hard

22.3.5 Writing and organizing Terraform with modules

Generally, Rackspace maintains modules for most common use cases, and uses these modules to build out your account. If we do not have a pre-existing module, the next best choice is to use the built-in `aws_*` resources offered by the AWS provider for Terraform. Please let us know if we don't have a module or best practice for building out a specific resource or AWS product.

A common recommendation in the Terraform community is to think of modules as functions that take an input and provide an output. Modules can be built in your shared repository or in your account repositories. If you find yourself writing the same set of resources or functionality over and over again, consider building a module instead.

When to consider writing a module:

- When multiple resources should always be used together (e.g. a CloudWatch Alarm and EC2 instance, for autorecovery)
- When Rackspace has a strong opinion that overrides default values for a resource
- When module re-use remains shallow (don't nest modules if at all possible)

22.3.6 General Terraform Style Guide

Terraform files should obey the syntax rules for the HashiCorp Configuration Language (HCL) and the general formatting guidelines provided by the Terraform Project through the `fmt` command.

In addition, Rackspace follows the following standards when writing Terraform:

- Use Snake Case for all resource names
- Declare all variables in variables.tf, including a description and type
- Declare all outputs in outputs.tf, including a description
- Pin all modules and providers to a specific version or tag
- Always use relative paths and the file() helper
- Prefer separate resources over inline blocks (e.g. aws_security_group_rule over aws_security_group)
- Always define AWS region as a variable when building modules
- Prefer variables.tf over terraform.tfvars to provide sensible defaults
- Terraform versions and provider versions should be pinned, as it's not possible to safely downgrade a state file once it has been used with a newer version of Terraform

22.3.7 Rackspace Terraform Module Standards

Rackspace maintains a number of Terraform modules available at https://github.com/rackspace-infrastructure-automation. Contributions should follow these guidelines.

- use semantic versioning for shared code and modules
- always point to GitHub releases (over a binary or master) when referencing external modules
- always extend, don't re-create resources manually

- parameters control counts, for non-standard numbers of subnets/AZs/etc.
- use overrides to implement Rackspace best practices
- use variables with good defaults for things Rackspace expects to configure
- Modules should use semantic versioning light (Major.minor.0) for AWS account repositories
- Modules should be built using the standard files: `main.tf, variables.tf, output.tf`
- Consider writing tests and examples, and shipping them in directories of the same name
- Readme files at should contain a description of the module as well as documentation of variables. An example of documentation can be found here.
- The files in `.circleci` are managed by Rackspace and should not be changed. If you would like to submit a module, please do so without this folder.
- The files in `example` can be named anything as long as they have `.tf` as the extension.
- The `tests` directory must be called `tests` and each test must be `test#``. Inside each *test#* folder should be exactly one file called `main.tf`
- Use Github's .gitignore contents for Terraform.

variables.tf

This file must include the following code block at the beginning or end of the file.

```
variable "environment" {
  description = "Application environment for which this network is being created. one␣
↪of: ('Development', 'Integration', 'PreProduction', 'Production', 'QA', 'Staging',
↪'Test')"
  type        = "string"
  default     = "Development"
}
 variable "tags" {
  description = "Custom tags to apply to all resources."
  type        = "map"
  default     = {}
}
```

main.tf

This file must include the following code block at the top of the file. Other variables can be added to this block.

```
locals {
  tags {
    Name            = "${var.name}"
    ServiceProvider = "Rackspace"
    Environment     = "${var.environment}"
  }
}
```

In any resource block that supports `tags` the following code should be used:

```
tags = "${merge(var.tags, local.tags)}"
```

This takes the tag values that are in `variable.tf` and combines them with any values defined in `main.tf` in the `locals` block.

Secrets storage using Terraform

Rackspace recommends storing secrets for Terraform using AWS KMS; embed ciphertext values as data sources in Terraform configurations. Here's some of the specifics and considerations:

- Use aws_kms_key to create a KMS key for use by Terraform; you should apply a key policy that allows IAM roles and users to use the key, because federated accounts can't access KMS keys using the default policy statements (e.g. most Rackers and Customers):

```
resource "aws_kms_key" "terraform_config" {
  description = "terraform_config"
  is_enabled  = true

  policy = <<EOF
  {
    "Version": "2012-10-17",
    "Id": "key-default-1",
    "Statement": [
      {
        "Sid": "Default IAM policy for KMS keys",
        "Effect": "Allow",
        "Principal": {
          "AWS": "arn:aws:iam::123456789012:root"
        },
        "Action": "kms:*",
        "Resource": "*"
      },
      {
        "Sid": "Enable IAM user to perform kms actions as well",
        "Effect": "Allow",
        "Principal": {
          "AWS": "${module.terraform_circleci_iam.circleci_user_arn}"
        },
        "Action": "kms:*",
        "Resource": "*"
      }
    ]
  }
EOF
}
```

- You will need to manually use the AWS CLI (and the key-id for the key you created in the previous step) to encrypt your secrets (mind any line endings if you use file:// to encrypt):

```
$ aws kms encrypt \
    --key-id 438290482-e36a-4803-a7d0-db436278 \
    --plaintext "super_secret" \
    --encryption-context resource=my_database,key=password \
    --output text --query CiphertextBlob
```

- Equipped with the ciphertext from the previous command, you can now use aws_kms_secrets to expose the secret as a data source for further use in Terraform:

```
data "aws_kms_secrets" "example" {
  secret {
    # ... potentially other configuration ...
    name    = "master_password"
    payload = "base64secret=="
```

(continues on next page)

(continued from previous page)

```
    context {
      resource = "db01"
      key      = "password"
    }
  }

  secret {
    # ... potentially other configuration ...
    name    = "master_username"
    payload = "base64secret=="

     context {
       resource = "db01"
       key      = "username"
     }
  }
}

resource "aws_rds_cluster" "my_database" {
  # ... other configuration ...
  master_password = "${data.aws_kms_secrets.example.plaintext["master_password"]}"
  master_username = "${data.aws_kms_secrets.example.plaintext["master_username"]}"
}
```

Note the use of context values; these are used as encryption context key pairs in KMS. These context values can be used by KMS to ensure a specific secret is always accompanied by the same context values (integrity), and may be emitted in CloudTrail logs or included in error messages (debugging).

22.4 Terraform Style Conventions

22.4.1 Key Concepts

Account Environment A collection of one or more layers defined within a single account repository. An account environment describes the complete infrastructure deployed for a single customer account.

Layer A layer is a logical grouping of related resources, data sources, and modules that should be managed together by Terraform. Layers are used to break complex environments into multiple logical subsets. Each layer should independently define configuration for all needed providers. Layers can represent different environments within an account, such as "production" and "test", different regions a customer application is hosted, or application tiers, like "database", "web server", or "network". When designing layers, the following considerations should be taken:

- Resources that are frequently modified together should be in the same layer, for example an EC2 instance, and its related IAM Role and policies should remain in a single layer.
- Smaller layers will limit blast radius and make Terraform state refreshes and updates quicker and safer.
- Dependencies between layers should always flow one way, taking 000base, 100data, and 200compute layers as an example, 000base should not reference anything in 100data or 200compute, and 100data should not reference anything in 200compute.
- Use a data source (*Code Structure*) to read another layer's state. Never write to another layer's state directly.

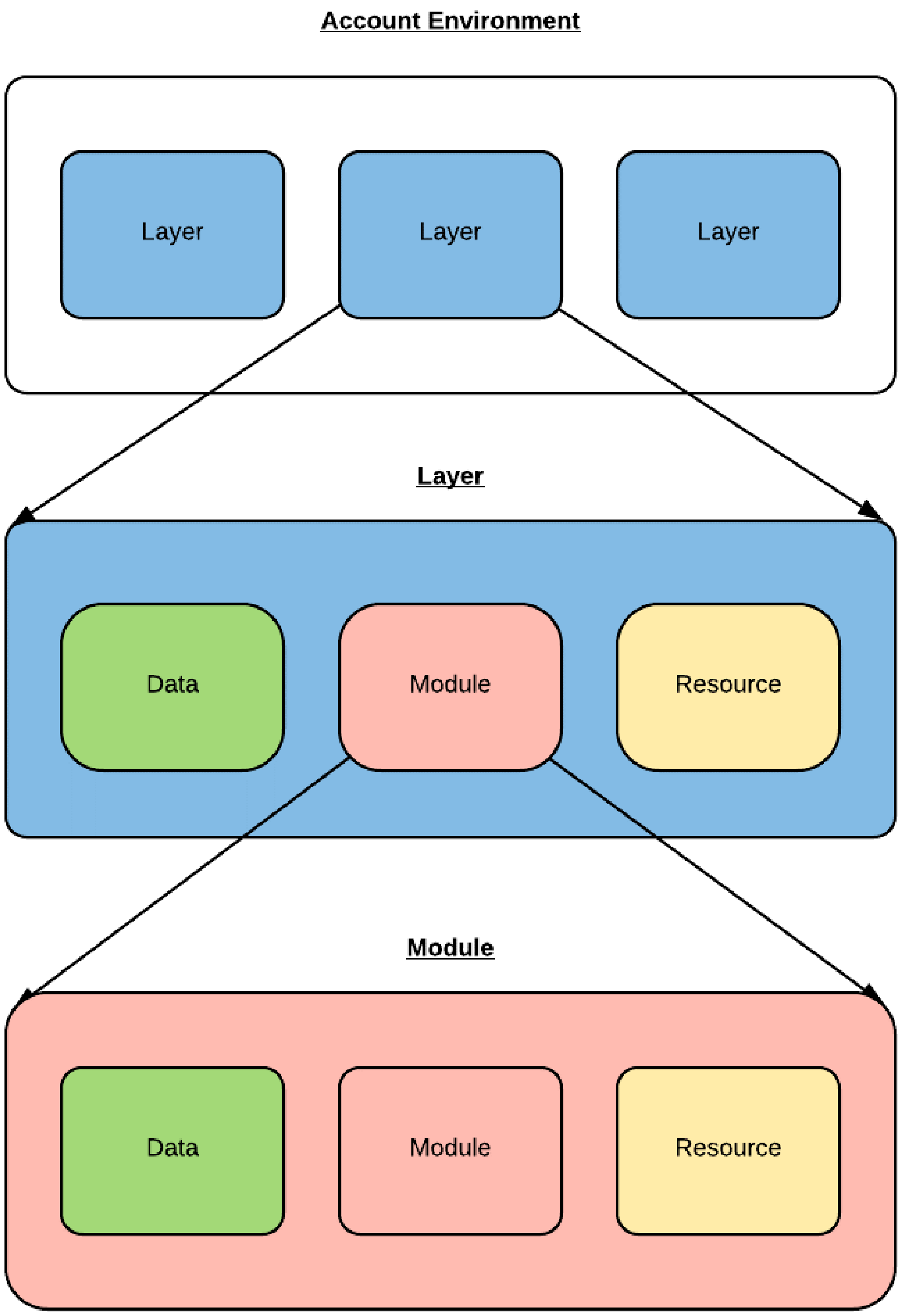
Module Module is a collection of connected resources which together perform the common action (for example, aws-terraform-vpc_basenetwork module creates VPC, subnets, NAT gateway, etc). It depends on provider configuration(s), which should normally be defined at a higher level. Rackspace maintains a number of modules, and customer specific modules can be created as needed.

Resource Resource is `aws_vpc`, `aws_db_instance`, and so on. Resource belongs to provider, accepts arguments, outputs attributes and has lifecycles. Resource can be created, retrieved, updated, and deleted.

Data Source Data source performs read-only operation and is dependent on provider configuration, it can be used in modules and layers to lookup information about the Account Environment.

Data source `terraform_remote_state` can be used to output from one layer to another (*Code Structure*).

The HTTP data source makes an HTTP GET request to the given URL and exports information about the response which is often useful to get information from endpoints where native Terraform provider does not exist.


Account Environment
Layer
Layer
Layer
Layer
Data
Module
Resource
Module
Data
Module
Resource

22.4.2 General Terraform Style Guide

Terraform files should obey the syntax rules for the HashiCorp Configuration Language (HCL) and the general formatting guidelines provided by the Terraform Project through the `fmt` command.

In addition, Rackspace follows the following standards when writing Terraform:

1. Use snake_case (lowercase with underscore character) for all Terraform resource or object names.
2. Variable defaults and arguments:
   - Declare all variable blocks for a module in **variables.tf**, including a description and type
   - When working within a layer: provide **no** defaults defined in **variables.tf** with all variable arguments being provided in `terraform.tfvars`
   - When working within a module: provide sensible defaults where appropriate; defaults can be empty string/list/map where variable is optional and being empty is handled gracefully
3. Declare all outputs in **outputs.tf**, including a description.
4. Pin all modules to a specific tag.
5. Pin all providers to major version. (for example, ‘‘‘~> 1.2: any non-beta version >= 1.2.0 and < 2.0.0, e.g. 1.X.Y‘‘‘).
6. Always use relative paths and the file() helper.
7. Prefer separate resources over inline blocks (for example, `aws_security_group_rule` over `aws_security_group`).
8. Always define AWS region in the provider block.
9. Terraform versions and provider versions should be pinned, as it’s not possible to safely downgrade a state file once it has been used with a newer version of Terraform.

22.4.3 Code Structure

Terraform Environment Standards

Customer account repositories should follow the layered style represented in the sample repository https://github.com/rackspace-infrastructure-automation/terraform-standards-examples/tree/master/example_3.1

Here are some considerations for building a new customer environment:

1. `_main` should be used only for initialisation. Resources should be added to newly created layers.
2. Unless there is a logical reason to deviate, the following default layers *should* be used. The idea behind the numbered prefixes is to deploy lower numbered layers first.
   1. `000base`: VPC, Endpoints, Route53 Internal Zone, SSM Service Role, SNS, Peering, VPN, Transit Gateway, Custom IAM, Directory Service
   2. `100data`: RDS, DynamoDB, Elasticache, S3, EFS, Elasticsearch
   3. `200compute`: EC2, LBs, SQS
3. Be sure to update the backend s3 key value in **main.tf** for each layer.
4. Security Groups should be defined within the layer in which the resource it is to be attached to resides. Take the following into consideration when defining security group rules:
   - If the `source_security_group_id` is in a previous layer, import via remote state

   - If the `source_security_group_id` is in the same layer, proceed as normal
   - If the `source_security_group_id` is in a following layer, the rule should be moved into the following layer alongside the source group once it is created

5. Leverage data source outputs to reference required information in another layer. For example, see Terraform Standards Examples.
6. **README.md** files *must* exist and describe the contents of each layer. An example of documentation can be found here for layer modules.

   `terraform-docs` is a tool to help create the documentation, and can found here.

Terraform Module Standards

Rackspace maintains a number of Terraform modules available at https://github.com/rackspace-infrastructure-automation . Contributions should follow these guidelines.

1. When a count is required, use of a variable is strongly recommended due to Terraform limitations.
2. When a variable value must be determined during execution, no default argument should be set, in all other cases a good default value should be included.
   - eg. ref required
3. Modules should use "semantic versioning" (major.minor.revision) for customer shared module repositories. Good release notes should be included.
4. Modules *must* include the following files, even if empty: **main.tf**, **variables.tf**, and **outputs.tf**.
   - Additional Terraform files can be included in order to logically separate resources into multiple files.
5. Modules *must* include an examples directory. If CICD testing is available, modules should contain a tests directory. Each distinct test or example should be placed in a descriptively named subdirectory. Subdirectory contents should meet all defined standards.

   For example, see the Terraform Standards Examples.
6. **README.md** files *must* exist and contain a description of the module as well as documentation of variables and outputs. An example of documentation can be found here for layer modules.

   terraform-docs is a tool to help create the documentation, and can found here. Version v0.6.0 of terraform-docs is used to generate documentation for all Rackspace managed modules.
7. The files in **.circleci** are managed by Rackspace and ***should not*** be changed. There is no requirement to modify files found in **.circleci** when adding an additional module.
8. Use Github's .gitignore contents for Terraform.

Getting started with structuring Terraform configurations

Refer back to the *Key Concepts* section if you are unsure what each Terraform structure is for.

Layout

The following diagram shows how the layer should be structured:

```
| layers/
| ├── _main
| │     ├── main.tf
| │     └── variables.tf
| ├── 000base
| │     ├── README.md
| │     ├── main.tf
| │     ├── outputs.tf
| │     ├── terraform.tfvars
| │     └── variables.tf
| └── 100data
| |     ├── README.md
| |     ├── main.tf
| |     ├── outputs.tf
| |     ├── terraform.tfvars
| |     └── variables.tf
| └── 200compute
|     ├── README.md
|     ├── main.tf
|     ├── outputs.tf
|     ├── terraform.tfvars
|     └── variables.tf
```

The following diagram shows how the modules should be structured:

```
| modules/
| ├── example
| │     └── main.tf
| ├── globals
| │     ├── main.tf
| │     └── outputs.tf
| └── s3_cf_website
|       ├── README.md
|       ├── examples
|       │     ├── main.tf
|       │     └── variables.tf
|       ├── main.tf
|       ├── outputs.tf
|       └── variables.tf
```

Resource and data source arguments

1. Resource names should be descriptive and avoid duplication of the resource type, where possible. Shorter resource names should be preferred over longer names, if both are descriptive. Duplication of resource type in part or whole is preferred over the use of non-descriptive names, such as this, that, or thing.

   - **Bad:** resource `aws_cloudwatch_log_group` "this" { - Non-descriptive resource name
   - **Good:** resource `aws_cloudwatch_log_group` "log_group" { - Descriptive resource name
   - **Best:** resource `aws_cloudwatch_log_group` "apache" { - Descriptive resource name without duplication of resource type.

2. Using singular nouns for names is preferred. If an individual resource is commonly referred to in the plural (eg, logs when referring to a CloudWatch Log Group), then a plural noun is acceptable.

   - **Good:** file

- **Good:** bucket
- **Good:** logs
- **Bad:** files

3. Include count argument inside resource blocks as the first argument at the top and separate by newline after it.
   - Good example
   - Bad example
4. Any **resource property** that **requires multiple lines** should fall below all properties that can be defined on a single line. Each **resource property** that requires multiple lines should have **blank lines** between itself and any other property.
   - Good example
   - Bad example
5. When present, `depends_on` and `lifecycle` should be the last two resource properties defined respectively. Each should be separated by a single blank line.
   - Good example
   - Bad example
6. Boolean values should not be used to directly set the value in count. Instead, a condition should be used.
   - **Bad:** `count = "${var.create_public_subnets}"`
   - **Good:** `count = "${var.create_public_subnets ? 1 : 0}"`
   - **Good:** `count = "${var.disable_nat_gateway ? 0 : 1}"`

Example Terraform Files

- main.tf: call modules, locals and data-sources to create all resources
- variables.tf: contains declarations of variables used in **main.tf**
- outputs.tf: contains outputs from the resources and modules created in **main.tf**
- terraform.tfvars: should only be used in layers.
- README.md: description of layer or module, including variables and outputs.

22.4.4 Secret storage using Terraform

Irrelevant of the strategy used to manage the creation and/or usage of passwords in Terraform it is important to understand how these are stored once they are used. Whether you hard code a secret (never to be done), create it with the Random provider, or decrypt a KMS encrypted string, the result is that this secret will always be visible in plaintext in the state file. It is therefore the state file that needs protecting. It is for this reason that we use remote state backends within our MIAC models where the storage location can ensure that the state files are encrypted, and the storage mechanism locked down to only those that should have access. When working with console managed customers (AWS primarily) the working practice is to create an AWS S3 bucket to still use an encrypted remote state backend, but with a lifecycle policy of 30 days so after this time the state will no longer exist.

It is the state containing these passwords in plaintext - as well as generally being a very poor, unmanageable, and non-scaleable option - that makes storing the state files along with the code a very bad idea.

Guidance

Secrets are typically going to fall into one of two categories: they exist and we need to use them, or they do not exist and we need to create and use them.

AWS has services built into the fabric that aid us in this endeavour. There is the AWS Systems Manager Parameter Store which has options to use KMS encrypted SecureStrings, and there is the AWS Secrets Manager. By storing the secrets in one of these services we can access them programmatically in code without needing to hard code them (big tick for clean code), the customer can add them to the console ahead of us using them if they already exist, or if we create them via Terraform we can store them in the console so we need never know them and the customer can retrieve them post deployment. In the case of AWS Secrets Manager it also opens the option of using automatic credential rotation.

If you are needing to create a password/secret (RDS password, AD password, token for CloudFront header, etc.) you can use the Terraform random provider: https://www.terraform.io/docs/providers/random/index.html

This is a basic use of the random provider to create a random string:

Secrets - Random String:

```
provider "random" {
  version = "~> 2.1"
}

resource "random_string" "rds_password" {
  length = 20
  lower = true
  upper = true
  number = true
  special = false
}
```

This example will give us a 20 character string containing upper- and lowercase alphanumerical characters. You can then use the output of this in other resources including the password argument of a RDS module call, or the value of an AWS SSM Parameter Store parameter. The next example shows creating, storing, and using the password (shortened for brevity).

Secrets - Random String Store and Use:

```
provider "aws" {
  version = "~> 2.20"
}

provider "random" {
  version = "~> 2.1"
}

resource "random_string" "rds_password" {
  length = 20
  lower = true
  upper = true
  number = true
  special = false
}

resource "aws_ssm_parameter" "rds_password" {
  name = "${lower(var.environment)}-rds-password"
  type = "SecureString"
  value = "${random_string.rds_password.result}"
```

(continues on next page)

(continued from previous page)

```
  tags = "${local.tags}"
}

module "rds" {
  source = "git@github.com:rackspace-infrastructure-automation/aws-terraform-rds//?
↪ref=v0.0.11"

[..]
password = "${random_string.rds_password.result}"
[..]

}
```

This example takes our random string and adds it to an AWS SSM parameter as a SecureString and then uses it as the input to the password argument in the RDS module. The same pattern would work for a password for Active Directory. The same pattern would work for adding a header to a CloudFront distribution that must be injected to allow traffic to a backend website bucket. It is flexible and secure and does not require you to work outside of Terraform code to implement.

This is a clean method for creating random strings and you can check the provider documentation if you wanted to make the strings more secure by changing length, adding symbols, etc.

Some in the security community would recommend using random words rather than more traditional patterns; for completeness here is an example of that:

Secrets - Random Pet:

```
$ cat example.tf

provider "random" {
  version = "~> 2.1"
}

resource "random_pet" "pet" {
  count = 5

  length = "${count.index + 1}"
  separator = ""
}

output "pets" {
  value = "${random_pet.pet.*.id}"
}

$ terraform output
pets = [
katydid,
summaryliger,
mainlyexcitinggrubworm,
merelygentlysteadycub,
openlypresumablylikelyblessedpeacock
]
```

There could be occurrences where a customer wants to provide a password or wants a specific pattern that we can't capture in code (because, as we have said, that's bad code).

As well as creating resources in AWS SSM Parameter Store we can also pull values from the store as well. Using our RDS example, here we can pull the password value and pass this into our RDS module:

Secrets - Data SSM Parameter:

```
provider "aws" {
  version = "~> 2.20"
}

data "aws_ssm_parameter" "rds_password" {
  name = "customer-provided-rds-password"
}
module "rds" {
  source = "git@github.com:rackspace-infrastructure-automation/aws-terraform-rds//?
↪ref=v0.0.11"

[..]
password = "${data.aws_ssm_parameter.rds_password.value}"
[..]
}
```

The examples so far have focused on AWS SSM Parameter store but we can also use AWS Secrets Manager. In the next two examples we use an existing secret, and we store a new secret. In the first example we are using the current version of the person and we look up the password by the name which works where the secret is in the same account and region, otherwise you need to use the arn argument as documented here: https://www.terraform.io/docs/providers/aws/d/secretsmanager_secret.html .

Secrets - Data Secrets Manager:

```
provider "aws" {
  version = "~> 2.20"
}

data "aws_secretsmanager_secret" "rds_password" {
  name = "customer-provided-rds-password"
}

data "aws_secretsmanager_secret_version" "rds_password" {
  secret_id = "${data.aws_secretsmanager_secret.rds_password.id}"
}

module "rds" {
  source = "git@github.com:rackspace-infrastructure-automation/aws-terraform-rds//?
↪ref=v0.0.11"

[..]
password = "${data.aws_secretsmanager_secret_version.rds_password.secret_string}"
[..]
}
```

Secrets - Secrets Manager Store and Use:

```
provider "aws" {
  version = "~> 2.20"
}

provider "random" {
  version = "~> 2.1"
}

resource "random_string" "rds_password" {
```

(continues on next page)

(continued from previous page)

```
  length = 20
  lower = true
  upper = true
  number = true
  special = false
}

resource "aws_secretsmanager_secret" "rds_password" {
  name                    = "${lower(var.environment)}-rds-password"
  recovery_window_in_days = 7

  tags = "${local.tags}"
}

resource "aws_secretsmanager_secret_version" "rds_password" {
  secret_id = "${aws_secretsmanager_secret.rds_password.id}"
  secret_string = "${random_string.rds_password.result}"
}

module "rds" {
  source = "git@github.com:rackspace-infrastructure-automation/aws-terraform-rds//?
↪ref=v0.0.11"

  [..]
  password = "${random_string.rds_password.result}"
  [..]
}
```

Deprecated Guidance

Note: The following information was the guidance given to customers and Rackers in the original Phoenix documentation. While this is still a valid solution it is cumbersome for all involved. The information in the above subsection should be considered the preferred route to take when dealing with secrets.

Rackspace recommends storing secrets for Terraform using AWS KMS; embed ciphertext values as data sources in Terraform configurations. Here's some of the specifics and considerations:

- Use ***aws_kms_key*** to create a KMS key for use by Terraform; you should apply a key policy that allows IAM roles and users to use the key, because federated accounts can't access KMS keys using the default policy statements (e.g. most Rackers and Customers):

Example aws_kms_key:

```
resource "aws_kms_key" "terraform_config" {
  description = "terraform_config"
  is_enabled = true

  policy = <<EOF
  {
    "Version": "2012-10-17",
    "Id": "key-default-1",
    "Statement": [
      {
        "Sid": "Default IAM policy for KMS keys",
```

(continues on next page)

(continued from previous page)

```
        "Effect": "Allow",
        "Principal": {
          "AWS": "arn:aws:iam::123456789012:root"
        },
        "Action": "kms:"*",
        "Resource": "*"
      },
      {
        "Sid": "Enable IAM user to perform kms actions as well",
        "Effect": "Allow",
        "Principal": {
          "AWS": "${module.terraform_circleci_iam.circleci_user_arn}"
        },
        "Action": "kms:*",
        "Resource": "*"
      }
    ]
  }
EOF
}
```

You must manually use the AWS CLI (and the key-id for the key you created in the previous step) to encrypt your secrets (mind any line endings if you use `file://` to encrypt):

Example aws kms encrypt:

```
$ aws kms encrypt \
    --key-id 438290482-e36a-4803-a7d0-db436278 \
    --plaintext "super_secret" \
    --encryption-context resource=my_database,key=password \
    --output text --query CiphertextBlob
```

Equipped with the ciphertext from the previous command, you can now use aws_kms_secrets to expose the secret as a data source for further use in Terraform.

Example aws_kms_secrets

data "aws_kms_secrets" "example" {

secret { # … potentially other configuration … name = "master_password" payload = "base64secret=="

context { resource = "db01" key = "password"

}

}

secret { # … potentially other configuration … name = "master_username" payload = "base64secret=="

context { resource = "db01" key = "username"

}

}

}

resource "aws_rds_cluster" "my_database" { # … other configuration … master_password = "${data.aws_kms_secrets.example.plaintext["master_password"]}" master_username =

“${data.aws_kms_secrets.example.plaintext[“master_username”]}”

}

Note the use of context values; these are used as encryption context key pairs in KMS. These context values can be used by KMS to ensure a specific secret is always accompanied by the same context values (integrity), and may be emitted in CloudTrail logs or included in error messages (debugging).

Reference Resources

Source	URL
Fanatical Support for AWS Product Guide	https://manage.rackspace.com/aws/docs/product-guide/miac/using-terraform.html#general-terraform-style-guide
Terraform Best Practices	https://www.terraform-best-practices.com/

22.5 Using CircleCI

Rackspace employs continuous integration and delivery (CI/CD) to deploy your infrastructure based on the `master` branch of your repository. Therefore, the *master* branch should always reflect a consistent, deployable, and current expected state of your infrastructure. The `master` branch is also protected from unreviewed changes, and should only be modified through a pull request process.

Any user with GitHub repository access (see :ref:’using_github’) has the same access to CircleCI projects configured to build that repository, using the, “Log In with GitHub” button. These projects, as well as logs, artifacts, and configuration, can be found by directly visiting the `rackspace-infrastructure-automation` organization in CircleCI. You may also navigate to individual jobs and workflow steps by following the links from the CircleCI status checks on a pull request in GitHub.

22.5.1 Build Environment

Rackspace maintains a standard Docker container, `rackspace-toolbox`, that is used in every CircleCI job when building, testing, and deploying your repository. This container includes standard tools like tfenv, awscli, python, jq. It also contains our open source Terraform linting tool tuvok and our tooling that wraps Terraform commands, analyzes layers, and retrieves temporary credentials. All changes to this container and scripts are heavily tested before release. This container also gives us a mechanism to release product improvements and security fixes without impacting your repository or workflow. While we don’t publish the source Dockerfile for this container, the container itself is available from rackautomation/rackspace-toolbox to use locally. The container is versioned semantically, and your build configuration references the container with a tag like `rackautomation/rackspace-toolbox:1`.

Running Locally

Combined with your repository’s `.circleci/config.yml` file, that calls the various scripts in the toolbox, you are able to run the same builds locally, as long as the container contains AWS credentials (try running `aws configure list` to confirm), and you’ve defined the following environment variables:

```
TF_STATE_REGION='us-west-2',
TF_STATE_BUCKET='your-bucket-name'
TF_VAR_aws_account_id='your-aws-account-number'
```

As mentioned in *Using Terraform*, we recommend using an existing build's output to find out your state bucket name. You may also copy and paste the final commands from those builds, bypassing the need to set some environment variables.

22.5.2 Workflow Execution

Caution: This section refers to the CircleCI concepts of workflows, jobs, and steps. These should not be confused with the more general usage of terms like "workflow" or other services where these terms exist, like GitHub.

Job Steps and Configuration

On every branch, your builds run standard commands like `terraform fmt` and `terraform init`, as well as our custom linting tool, tuvok. We also calculate what layers have changed using a detailed comparison between the current branch and the most recent successful master build.

1. On branch `master`, your builds run both `terraform plan` and `terraform apply` for any changed or removed layers.
2. On any other branch, your builds will only run `terraform plan` for any changed or removed layers.

Your CircleCI configuration is stored in your repository at `.circleci/config.yml`. This file configures the CI/CD system with specific workflow steps to execute, and a defines a container in which to run them. This file should not be modified except through coordination with Rackspace.

In your CircleCI configuration, these two main steps are organized as two steps in the workflow, called "plan" and "apply", and we use the branch filtering functionality to ensure the "apply" step only runs on master, and depends on the plan step. If you're running terraform locally, it's important to use the same steps used in CI/CD, and also to never apply a change before it's been merged into master.

Failed Builds

From time to time, either due to code changes or Terraform/AWS internals, build steps may fail. For branches other than master, this is a totally normal part of writing and testing code. We also enforce that branches are up to date with the latest commits from master before merging, which can cause failures or plans that show unexpected changes. Most failed builds can be resolved through re-running the build or pushing additional commits or pulling in the latest changes from master.

In the event that a master build fails, on the plan or apply steps, Rackspace will automatically create an urgent, public ticket and investigate. We believe that these events could indicate larger problems with your infrastructure, partially applied changes to your infrastructure, or other impactful situations that require immediate attention.

22.5.3 Artifacts and Storage

Each build step writes logs in order to help record what is happening, make troubleshooting easier, and shed light on unexpected failures. These log files are made available in the CircleCI UI for a given build, in an "Artifacts" tab. Over time, we plan to make more artifacts available, with additional helpful output, to make it easier to troubleshoot problems or understand what's happening in a build. Please note that these artifacts eventually do expire.

22.5.4 AWS Credentials

Temporary credentials are issued to each build, at the start of the build, after our automation confirms that the build is associated with the correct GitHub repository, AWS account, and Rackspace customer. This confirmation relies on the same RSA keys used by GitHub to authorize access to the repository itself.

CircleCI builds now use temporary credentials rather than requiring an IAM user to be assigned to each AWS account. This is based on the same AWS STS-based temporary credentials mechanism used when you access your AWS account via the Fanatical Support for AWS control panel "Console" button.

There are several advantages to using temporary, short-lived credentials over static credentials, given how permissive and powerful Terraform's credentials need to be when building and managing AWS services on your account:

- Credentials are not stored permanently in the account or in Terraform state (an S3 bucket)
- Temporary credentials will be valid for a short period of time from the start of a build and expire safely
- Static credentials require a terraform change, slowing down the provisioning process for new accounts

22.6 Deploying Code

From time to time, you will be faced with a decision about how to deploy a file, instance configuration, agents, or even entire applications into an AWS environment (we refer to these collectively as 'artifacts' below). This page is intended to be our Fanatical AWS "best practices" and opinions around each option.

Many of these options provide an initial bootstrapping step that can then be used to perform additional work or copy additional files from other locations.

22.6.1 Bake files into AMI

Copy needed artifacts to an EC2 image, create an Amazon machine image (AMI), and rotate instances using the new AMI.

Variations:

- Create an instance manually and create an AMI
- Use tools like Packer to automatically build AMIs

Benefits:

- No dependence on external sources
- Fastest instance boot time
- Guaranteed state
- Strongly version controlled
- Can release updates to multiple files simultaneously
- Allows for testing before deployment
- Encryption possible

Drawbacks:

- Updates require a new AMI be built
- Storage cost of AMIs
- May have many AMIs with duplicate artifacts (e.g. one AMI per region)

- Requires additional management to clean up old AMIs
- Some artifacts need to be staged before AMI taken (i.e. sysprep style)
- Some artifacts (e.g. agents) may conflict with user-data/cloud-init

22.6.2 AWS CodeDeploy

Install CodeDeploy agent using one of the methods in this table, then use APIs to drive deployment from GitHub, S3, or another supported Git hosting provider. Embed additional artifacts in the deployment artifact or fetch them during CodeDeploy lifecycle hook execution.

Variations:

- Store artifacts in git and deploy them as part of application
- Fetch artifacts in S3 or another location

Benefits:

- CodeDeploy offers blue-green and canary deployment options
- Allows for live deployments of configuration files without rotating AMIs
- Config files can be tied to application deployments
- Encryption possible with S3

Drawbacks:

- Another agent to maintain and update
- Secrets should not be stored in git repositories
- Requires a deploy of application to update configuration files

22.6.3 User Data

Embed scripts in user-data or in cloud-init (called by user data). These scripts can fetch artifacts from S3, embed smaller artifacts directly, or even add package repositories and install packages.

Variations:

- Fetch files in S3 on boot
- Write files directly on boot

Benefits:

- Fastest ability to update files (just build more instances)
- Ability to always download the latest versions (e.g. for agents)
- Customer can deploy secrets without Rackspace needing to be involved
- S3 can be secured and accessible only from a VPC Endpoint
- Encryption possible

Drawbacks:

- Slows down provisioning of new instances
- Dependence on external source to boot instance
- Difficult / slow to debug, as autoscaling may terminate unhealthy instances

- No guarantee that instances will have same artifact versions
- Security controls must be managed appropriately, with 3rd party code without review on new versions

22.6.4 Native AWS APIs

Write your application to utilize native EC2 SSM Parameter Store or other AWS storage services to directly retrieve artifacts at runtime.

Variations:

- EC2 SSM Parameter Store
- S3, DynamoDB, etc

Benefits:

- No need to manage config file deployment
- Applications can be dynamically updated

Drawbacks:

- Significantly more opaque than a file-based method
- Requires a very good knowledge of the AWS APIs
- May require a re-architecture of the application

22.6.5 Terraform Module

Terraform offers a template provider that can be used to embed artifacts inline or as separate files. You can then use these artifacts via data sources when building a cloud-init configuration.

Variations:

- Keep files in git
- Embed scripts directly in Terraform file

Benefits:

- Terraform native functionality to deploy files with dynamic values
- Allows for collaboration on artifacts in Git
- Uses cloud-init for cross-platform functionality
- Less files strewn around various places
- Easier to pull in dynamic values from other APIs
- Single place to manage instance configuration

Drawbacks:

- Requires co-locating configuration files and Terraform files
- Inline artifacts make Terraform harder to read; painful escaping of strings
- Not in the spirit of Terraform or Infrastructure as Code
- Cannot dynamically update artifacts without applying Terraform again

22.7 Frequently Asked Questions

Should I have a backup of my GitHub repositories? Does Rackspace keep a backup of them as well?

GitHub keeps three copies of all data; we believe that these assurances about data reliability are adequate. Beyond that, we also have a copy of the Terraform files for every build that runs through CircleCI, as well as local copies of all revisions anywhere that someone has checked out a local copy recently. We are confident this is enough redundancy to keep your infrastructure safe, however GitHub does offer some tooling suggestions if you would like to create an additional backup of your data on your own.

Why am I seeing Terraform plans with EC2 instances that have one security group removed?

This is most likely from Passport. You should close the Passport request before making infrastructure changes, or restore the removed security group once the change has been applied.

I've submitted a pull request to my repository. Why isn't it being tested and checked by the continuous automation?

This most commonly occurs if you've created a fork of your repository and you're submitting a pull request from that fork instead of a branch on the same repository. In order to protect against the exfiltration of build data, our CI system won't build from a fork. Please push your forked branch to a branch on the original repository.

I'm seeing authentication failures in CloudTrail and in Compass when Terraform runs. What are these and what can I do about them?

Unfortunately, the `DescribeVpcClassicLinkDnsSupport` API call, used by the Terraform provider for AWS to determine EC2 capabilities, is recorded as an authentication failure when an AWS account does not have "EC2 Classic" enabled (a previous generation of the EC2 service). When this occurs, CloudTrail records an error message `"This request has been administratively disabled"`.

How do you handle multiple, concurrent changes? Won't they break on each other?

Pull requests are designed to fail testing when the branch is out of date with master. This ensures we'll only apply one set of Terraform changes at a time. We believe this provides adequate concurrency limits to prevent problems.

How can I share modules across repositories but still develop them quickly, without multiple pull requests for every change?

We recommend developing a module in the repository that will be first using it, to limit the amount of review and pull requests while you're in the development phase and iterating quickly. Once a module has become relatively stable, it can be relocated to your Shared Repository and used in multiple accounts.

How do I protect resources from accidental deletion?

Some Terraform-created resources should be protected from deletion, either due to external dependencies or because they contain data that shouldn't ever be destroyed. Likewise, sometimes a new resource should be created before the older one is removed, so that there's always a resource available to serve requests. For these use cases, we recommend using lifecycle blocks on resources to ensure they are never deleted, or are re-created in a specific order.

How do I tell Terraform to ignore my application deployment pipeline's changes? Other types of third-party changes?

For various reasons, you may want to ignore certain attributes on a resource. You may be updating a version attribute as part of a separate application deployment pipeline, or using a third-party service that manages a specific resource in AWS. For these use cases, we recommend using the ignore_changes attribute on specific resources. This *should be used sparingly*, as it has associated risks and makes your infrastructure harder to rebuild in a repeatable way.

Why don't you use Terraform's workspaces feature?

According to the Terraform project developers, workspaces are intended to be, "*temporary* copies of an infrastructure during development," modeled after git branches. "Most teams should not use workspaces, and should instead use the module-per-environment pattern." The developers intend to revise the *When To Use Workspaces* section to be a lot more explicit about what workspaces are suited for and what they are less suited for. In the mean time, they recommend users should instead adopt the module-per-environment pattern. This is what we've done with the layers concept.

How do I reapply the master branch on my account, without making infrastructure changes?

The most straightforward way to do this is by making a simple whitespace/comment change to any layers you'd like to have re-applied. Please reach out to your Support team if you need further assistance. At the current time, Rackspace does not monitor for outstanding/unapplied changes between your Terraform configuration and your infrastructure.

Can I take my Terraform files and scripts with me if I leave Rackspace?

All Terraform files used to build your infrastructure can be downloaded and taken with you at any time. Your Terraform module files are located in the GitHub repository connected to your AWS account. The Rackspace developed and managed Terraform module files, which are referenced by your Terraform module(s), are publicly available on a Rackspace owned GitHub repository and are licensed under the MIT license. If you would like assistance in cloning your GitHub repository or downloading your Terraform files, please open a ticket with us.

What is the IAM user on my account being used for? How can I be sure it is secure?

We're allocating an IAM user on every AWS account strictly to be used by Terraform when running through the CI system. This user's access keys are known only to the CI system. We will eventually remove this user and use a new process that relies on API keys, scoped only to your AW account, that can be used by the CI system to get temporary, short-lived credentials that have access to build and manage resources in your AWS account.

How many users can I add to GitHub? How do I add them?

By default, we limit the number of users we can add to your GitHub repositories to fifteen (15). Please contact us if you need additional users added, or need to exceed that limit.

How can I request a pull request review from Rackspace?

Our tooling and automation automatically tracks pull requests that you create, and prioritizes their review to our Support Teams. Please give us a call if you have something you need urgent review on, and we'll review immediately with you.

How do I give feedback on the infrastructure-as-code product?

Please open a normal Rackspace support ticket and submit your feedback. Support Rackers will pass your feedback along to our product teams. Your feedback means a lot to us. Thank you.

CHAPTER TWENTYTHREE

SERVICE BLOCKS

Rackspace knows that our customers have varying needs at different stages of their cloud journey. That's why we provide a set of support offers that solve for your needs at any stage of your cloud lifecycle. Our services include architecture help, access to experts to solve your problems, security assistance, 24x7 management, cost governance, and other value-added services - all backed by AWS certified engineers and architects.

Our offers allow you to customize your experience with the ability to choose the service options to match your needs. These offers are described below. For additional detail about what is included in each service block, download our Fanatical Support for AWS with Rackspace Service Blocks Service Overview.

23.1 Platform Essentials

Platform Essentials is the entry ticket for Rackspace services. All other services are built on top. Platform Essentials includes:

- AWS Support powered by AWS Certified Rackers and backed by AWS Enterprise Support
- Named Account Manager for ongoing business needs
- Unified billing for all your Rackspace platforms and other Managed Public Cloud Accounts
- Access to the Fanatical Support for AWS Control Panel to manage your AWS accounts, your users, and their permissions
- Industry leading tooling that provides you better insight into your AWS environment:
  - Compass: AWS inventory, best practice checks, cost analytics, and cost optimization
  - Waypoint: Rackspace account summary to track spend and critical information regarding your AWS accounts
  - Logbook: Aggregation of your control plane logs across all AWS accounts

Platform Essentials customers receive 24x7 guidance support on their AWS accounts. Rackspace will respond to support requests submitted via tickets in the following timeframes:

- Urgent: Production System Outage / Significant Business Impact [60 Minute Response Time]
- High: Production System Impaired / Moderate Business Impact [4 Hour Response Time]
- Normal: Issues and Requests / Minimal Business Impact [12 Hour Response Time]
- Low: General Information, Questions, and Guidance [24 Hour Response Time]

All requests should be made directly to Rackspace and we will escalate to AWS, if needed.

23.2 Architect & Deploy

With the Rackspace Architect & Deploy service block, our experts apply best practices to design and deploy public cloud infrastructure that meets your business needs while minimizing costs, maximizing availability, security and performance, and enabling you to outsource ongoing management activities to Rackspace. The Architect & Deploy offer includes:

- A Onboarding Manager to coordinate end-to-end activities and project manage your AWS deployment
- A Solutions Architect to understand your requirements and create a high-level proposal document for your approval
- A Build Engineer who will build and deploy the environment as per the design document
- Design Document: A document describing the detailed solution design. This document will be shared, and your approval of the design is required prior to deploying the solution
- AWS Environment: The deployed solution running in AWS
- Configuration of Rackspace standard monitoring that is integrated with the Rackspace ticketing system

23.3 Discover & Enhance

With the Rackspace Discover & Enhance service block, our experts assess your existing AWS environment in order to identify areas for enhancement. Based on your input, our engineers apply best practices to update your public cloud infrastructure so that you can be confident that it will continue to meet your business needs while minimizing costs and maximizing availability, security and performance.

The Discover & Enhance service block is available for any AWS environment that is already supported by the Rackspace Platform Essentials service block. When you purchase this service block, you also have the option to add on the Manage & Operate service block in order to upgrade to a higher level of ongoing management for your AWS environment.

23.4 Manage & Operate

With tooling, automation, monitoring and 24x7 access to certified cloud specialists for day-to-day operational support and management, Manage & Operate allows you and your team to rest easy knowing Rackspace has your back. Manage & Operate includes access to additional tooling like Passport (instance access request control tool), Watchman (turns monitoring alerts to tickets for Rackers to address) and Instance Scheduler (configuration of custom start/stop schedules for EC2 and RDS instances). Your Rackspace technical support professionals will help you resolve issues quickly and effectively any day of the year, any time of the day. Manage & Operate includes:

- Named Account Manager to coordinate escalations, follow AWS technical issues through to resolution, and help focus on the AWS technical operations of the account
- Access to 24x7x365 Technical Operations staffed around the clock and around the globe to help when you experience an issue with your AWS infrastructure
- 24x7x365 management of your environment
- Operating System management
- Setup of AWS Infrastructure monitoring using AWS CloudWatch and respond to your monitoring alerts via Rackspace Watchman tool with response time SLAs up to 15 minutes
- Passport: secured bastion access to your virtual instances

- Watchman: creation of monitoring alarm tickets for your account
- *Instance Scheduler*: configuration of custom start/stop schedules for EC2 and RDS instances

In addition to the response time SLAs of Cloud Foundation, Manage & Operate customers have access to:

- Emergency: Business-Critical System Outage / Extreme Business Impact [15 Minute Response Time]

Architect & Deploy or Discover & Enhance is a pre-requisite for any customers entering in to Manage & Operate.

23.5 Complex Cloud Operations

As a business matures or their cloud spend increases, operating AWS can become more complex. Complex Cloud Operations will help you manage this complexity with Rackspace experts that have worked with other similarly complex cloud deployments. Whether you desire a deeper technical relationship to drive outcomes or need assistance handling your architecture's complexity, Complex Cloud Operations can assist.

Complex Cloud Operations is offered in four tiers of support:

- Bronze: Lead Cloud Engineer shared with 10 customers
- Silver: Lead Cloud Engineer shared with 4 customers
- Gold: Lead Cloud Engineer shared with 2 customers
- Platinum: Dedicated Lead Cloud Engineer

Rackspace will recommend a tier of support (Bronze, Silver, Gold or Platinum) based on customer complexity and requirements. Quarterly, customers will work with their resources to scope what available capabilities will be delivered based on level of commitment and customer requirements. Potential activities include:

- ITIL problem management of recurring incidents
- Architecture diagrams of existing infrastructure
- Creation/Maintenance of basic post-deployment infrastructure configuration management scripts
- Review recommendations around Security, Availability, Performance, and AWS Trusted Advisor with remediation plan
- Implement cost saving recommendations by terminating idle or unused resources, right-sizing resources, updating previous generation resources
- Participate in Customer Change Advisory Boards and Stand-Ups
- Training sessions on relevant public cloud topics
- Well-Architected Reviews on different parts of your deployment
- In-Depth Roadmap Reviews for Rackspace Offers and Cloud Products
- Big Data, Serverless, and Container experts

23.5.1 Additional Rackspace Service Blocks

Rackspace offers additional services that can be layered on top of your Fanatical Support for AWS Service Blocks to create a fully managed cloud solution for your business needs.

23.6 Managed Security - Proactive Detection & Response

With Rackspace Managed Security – Proactive Detection and Response, our security experts defend your business against advanced cyber threats with 24x7x365 support from our Security Operations Center (SOC). Proactive Detection and Response includes:

- 24x7x365 monitoring and detection from security analysts using best-of-breed, curated technology (host-based detection, network detection, and a security analytics platform)
- Proactive cyber hunting to scan for anomalous activity
- Real-time response and remediation of threats with pre-approved actions
- Auto-Discovery and agent deployment of new compute instances with real time agent health status and environment coverage map
- Weekly and monthly reporting to communicate observations, alerts, and action
- Access to the Rackspace security dashboard that shows top vulnerabilities, hunt missions, events, and environment coverage map

23.7 Managed Security - Compliance Assistance

With Compliance Assistance, our Managed Security compliance experts assist customers with defining, managing, and validating selected Governance Risk and Compliance (GRC) requirements. Compliance Assistance includes:

- Configuration Hardening Monitoring
- Patch Monitoring
- File Integrity Monitoring
- User Access Monitoring
- Monthly and ad hoc compliance reports

23.8 Application Managed Services

Your business success is tied to the applications you rely on — from servicing customers, to managing supply chains, to getting new products to market. Application Managed Services from Rackspace helps optimize your application environment, so you can deliver on your service level agreements, free up resources and reduce costs. Services include:

- Advanced Configuration and Optimization: System engineers will custom-tailor your applications to fit your individual environment, and optimize performance and cost
- Administration, Monitoring and Maintenance: Application specialists will reduce the burden on your staff by administering, maintaining and continuously monitoring your applications
- Data Services: Data specialists will apply best practices and automated technology to modernize your database footprint and maximize the utility of your data

Please talk to your Account Manager if you are interested in learning more about the service block offers.