**Revision and Signoff Sheet**

Customer Sample– Cloud Runbook

Abstract

Execution-level document specifying customer approach to deploying applications on Amazon Web Services. This is a living document detailing Customer current approach to architecture, provisioning, and operations on AWS.

Delivered by:

Professional Services

**Change Record**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Reference |
| 6-24-2016 | AWS ProServe | 1.0 |  |
|  |  |  |  |

**Reviewers**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Version Approved | Position | Date |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[1 Purpose 3](#_Toc454544113)

[2 Related Documents 3](#_Toc454544114)

[3 Operations Overview 4](#_Toc454544115)

[4 Platform Operations 5](#_Toc454544116)

[4.1 Tagging 5](#_Toc454544117)

[4.2 Application Service Level Classification 7](#_Toc454544118)

[4.3 Application Recovery Classification 7](#_Toc454544119)

[4.4 Patch Management 8](#_Toc454544120)

[4.5 Platform Logging 9](#_Toc454544121)

[4.6 Configuration Management 10](#_Toc454544122)

[5 Provisioning & Service Catalog 13](#_Toc454544123)

[5.1 Provisioning 13](#_Toc454544124)

[5.2 Service Catalog 15](#_Toc454544125)

[6 Availability Management 15](#_Toc454544126)

[6.1 Backup & Restore 15](#_Toc454544127)

[6.2 Pilot Light 16](#_Toc454544128)

[6.3 Warm Standby 17](#_Toc454544129)

[6.4 Multi-site (AZ) 17](#_Toc454544130)

[6.5 Other Availability Considerations 18](#_Toc454544131)

[7 Platform Lifecycle Management 19](#_Toc454544132)

[7.1 Application Architecture Patterns: 19](#_Toc454544133)

[7.2 Security Architecture Patterns 24](#_Toc454544134)

[7.3 AMI Lifecycle Management 25](#_Toc454544135)

[8 Event Monitoring 27](#_Toc454544136)

[9 License Management 29](#_Toc454544137)

[10 Network Architecture 29](#_Toc454544138)

[10.1 CloudFormation patterns 29](#_Toc454544139)

[11 Security 30](#_Toc454544140)

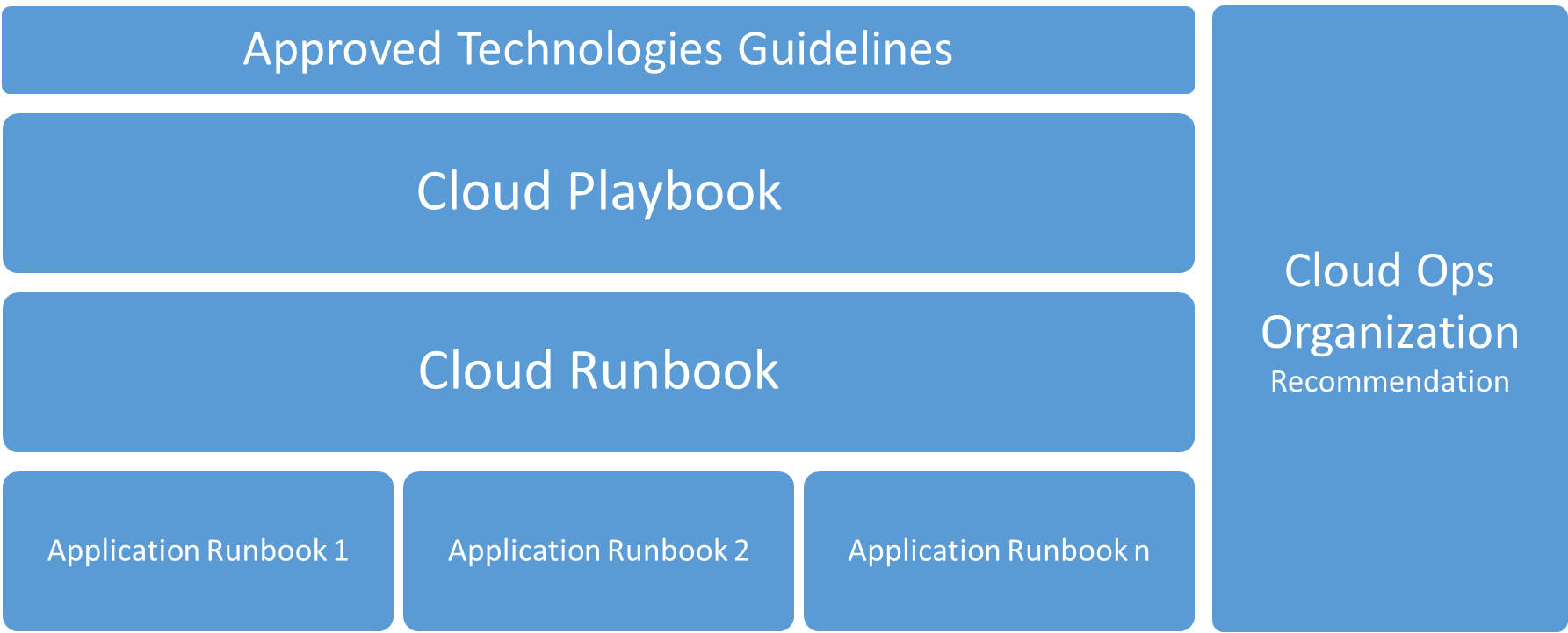
[12 Financial Management 30](#_Toc454544141)

# Purpose

This document should be used as a reference point for future choices and efforts in the operation of a cloud environment. It discusses many facets of the “how” and “why” of every day operations in the cloud and suggests steps that can be taken in the future. In many cases there are coded examples alongside of that suggestion, in some cases they will need to be created. While not specifically strategic, the intent of this document is to be used with a play book and it speaks to planned next steps in that document.

# Related Documents

The Cloud Runbook is part of a series of interrelated documents that address the topics of cloud adoption; including, technology selections, strategy formation, cloud execution, application-specific runbooks and even dealing with organizational considerations. The level of detail included in these documents is illustrated below, starting with the highest level strategy information at top, pushing down through the most specific and detailed at the bottom.



* Recommended Technologies Guidelines represent a framework to guide the organization in selection of technologies to insure an efficient use of resources, improve time to market and the ability to maintain service level operations.
* Cloud Playbook is a strategy-level document specifying CUSTOMER’s approach to deploying applications on and in a modern cloud architecture. Documents CUSTOMER’s current approach to architecture, provisioning, and operations in the cloud; and also provides the next steps to increase cloud adoption maturity.
* Cloud Runbook is an execution level document that provides the technical details of operating application and infrastructure in a cloud environment.
* Application Runbooks provide a view into application specific exceptions from the standard cloud guidelines as outlined in the Cloud Runbook.
* Cloud Team Organization outlines the resources, skills and teams that are involved in driving the cloud adoption, a Cloud Center of Excellence (COE) as well as those who manage the day to day provisioning, usage, monitoring, and cost of cloud resources (Cloud Engineering).

# Operations Overview

As organizations adopt cloud services, the organization must transform in order to fully realize the benefits of cloud technologies. Operationalizing your cloud service offering includes adapting the services offered, the manner in which they are offered as well as how they are offered. This document is to be a living document that outlines the IT service management features like proactive monitoring, patch, business continuity, and financial management, and processes like change and incident management. There are options identified that can be implemented dependent upon the maturity of your organization.

To be optimal, service management and operational functions must be proactive and supported by automation, as opposed to reactive and supported by manual human intervention. This applies both to deploying resources and automating responses to potential issues by designing for failure. Repetitive manual tasks should be reduced through the use of automation to allow operations teams to focus on value-add work. Operational areas that are important to address early in the cloud journey include Financial Management, Platform Operations, Monitoring & Incident Resolution, Provisioning & Service Catalog, Availability Management and Application Lifecycle Management. Similar to the iterative approach of agile development, there will be an iterative approach to cloud operations.

The specific categories are as follows:

|  |  |
| --- | --- |
| Operations Category | Sub-categories |
| Platform Operations | Tagging  Application Service Level Classification  Application Recovery Classification  Patch Management  Platform Logging |
| Provisioning & Service Catalog | Infrastructure Provisioning  Service Catalog |
| Availability Management | Backup & Recovery (DR)  High Availability |
| Platform Lifecycle Management | Application Architecture Patterns  Security Group Patterns  AMI Lifecycle Management  Platform Monitoring  License Management  Network Architecture Patterns  Load Balancers  Firewall Management  Application Logging  Resource Cost Optimization  Configuration Management |
| Financial Management | Chargeback Process  Billing Method  Budgeting and Forecasting  Reporting |

# Platform Operations

## Tagging

Description

Tagging is the most important documentation you can do in cloud and enables the strategies discussed further in this document. While each section discusses specific items the constant theme is tagging. Tagging is the practice of adding information to an instance or virtual machine. Each tag should have a specific function and it is important that they have the same name and the values are the same across the instances. This allows for searching, scripting and alerts. These function will make up the backbone of your automation and allow a small number of administrators to manage all of the systems.

Technical Approach

Tags are key value pairs that consist of Key Names (categories) and Key Values (the actual names of the tags). The Key Names identified below will be used. The Key Values are located in the corresponding Runbook categories found in the following sections.

|  |
| --- |
| ServiceLevel |
| RecoveryTier |
| ApplicationID |
| EnvironmentLevel |
| CostCenter |
| MedPharma |
| BackupPolicy |

Enforcement of tags will be handled through the following approaches:

* **CloudFormation-** Tagging can be enforced as part of the initial parameters and constraints, in each script. These constraints will fail a script if tags are missing.



This style of Coding turns into a cloud formation section looking like this:

"Tags":[  
 {"Key" : "Application", "Value": {"Ref" : "TagsApplicationParam"}},  
 {"Key" : "CostCenter", "Value": {"Ref" : "TagsCostCenterParam"}},  
 {"Key" : "Environment", "Value": {"Ref" : "TagsEnvironmentParam"}},  
 {"Key" : "Backup", "Value": {"Ref" : "TagsBackupParam"}},  
 {"Key" : "ServiceClass", "Value": {"Ref" : "TagsServiceClassParam"}},  
 {"Key" : "RecoveryTier", "Value": {"Ref" : "TagsRecoveryTierParam"}}  
]

* **Using the AWS Console-**
* IAM can control CloudFormation access, but the user will always be able to create a single machine via the EC2 control panel. We can enforce tagging on these devices via Lambda scripting and scheduled procedures.
* Lambda Scripting:
  + Located at: %CodeCommit%/LambdaFunctions/MissingTag.py
* These tags can be placed on most item in the AWS environments:

|  |  |  |
| --- | --- | --- |
| Resource | Tagging support | Tagging restrictions |
| AMI | Yes | None |
| Bundle task | No |  |
| Customer gateway | Yes | None |
| Dedicated Host | No |  |
| DHCP option | Yes | None |
| EBS volume | Yes | None |
| Instance store volume | No |  |
| Elastic IP | No |  |
| Instance | Yes | None |
| Internet gateway | Yes | None |
| Key pair | No |  |
| NAT gateway | No |  |
| Network ACL | Yes | None |
| Network interface | Yes | None |
| Placement group | No |  |
| Reserved Instance | Yes | None |
| Reserved Instance listing | No |  |
| Route table | Yes | None |
| Spot Instance request | Yes | None |
| Security group - EC2-Classic | Yes | None |
| Security group - VPC | Yes | None |
| Snapshot | Yes | None |
| Subnet | Yes | None |
| Virtual private gateway | Yes | None |
| VPC | Yes | None |
| VPC endpoint | No |  |
| VPC flow log | No |  |
| VPC peering connection | Yes | None |
| VPN connection | Yes | None |

## Application Service Level Classification

Description

This tag expresses the service level of the specific application or portions of the application. This value combined with the RecoveryTier tag, describes the architecture choices advisable for a new application and can be included in external scripting to verify that systems are running correctly and changes that occur have not pushed an application outside of the designed tolerances.

|  |  |
| --- | --- |
| Key Name | Key Value |
| ServiceLevel | Class5 |
|  | Class4 |
|  | Class3 |
|  | Class2 |
|  | Class1 |

Availability Patterns (detailed in sections below) associated with each service class:

|  |  |
| --- | --- |
| Service Class | Availability Pattern Associated |
| Service Class 5 | Warm-standby (in multi-AZ), Multi-site (in multi-AZ), or Multi-region |
| Service Class 4 | Warm-standby (in multi-AZ) or Multi-site (in multi-AZ) |
| Service Class 3 | Backup & Restore single instances with Snapshots stored in S3 |
| Service Class 2 | Backup & Restore single instances with Snapshots stored in S3 |
| Service Class 1 | Backup & Restore single instances with Snapshots stored in S3 |

## Application Recovery Classification

Description

The RecoveryTier tag identifies the backup strategy as well as the availability pattern for the application. This tag is also expressed in BackupPolicy, discussed later. A tag of RecoveryTier: Tier1 should generate the most aggressive backup and life cycle policy, generating a snapshot at least every 4 hours.

|  |  |
| --- | --- |
| Key Name | Key Value |
| RecoveryTier | Tier1 |
|  | Tier2 |
|  | Tier3 |
|  | Tier4 |
|  | Tier5 |

Availability Patterns (detailed in sections below) associated with each service class:

|  |  |  |
| --- | --- | --- |
| Recovery Tier | RTO/RPO (in hours) | Availability Pattern Associated |
| Recovery Tier 1 | 24/4 | Warm-standby (in multi-AZ), Multi-site (in multi-AZ), or Multi-region |
| Recovery Tier 2 | 48/24 | Warm-standby (in multi-AZ) or Multi-site (in multi-AZ) |
| Recovery Tier 3 | 72/24 | Backup & Restore single instances with Snapshots stored in S3 |
| Recovery Tier 4 | 168/24 | Backup & Restore single instances with Snapshots stored in S3 |
| Recovery Tier 5 | 169+/24 | Backup & Restore single instances with Snapshots stored in S3 |

## Patch Management

Description

Patching in the cloud is similar to the patch process in your current environment. Making backups and snapshots of systems before a patch is applied is far simpler and more convenient.

Current CUSTOMER Policy

* All systems must have an identified maintenance window
* All patches will be delivered within 31 days of availability
* All patching tasks are automated
* Testing should be automated (or app teams automatically notified of completion)
* Non prod environments should be patched prior to prod
* Before a system is patched, a snapshot should be made of it.

Tagging:

* These two tags allow for automation scripts to be used to interact with you instances at the application level and control the implementation of the patches.
* Controlling what environment level and which application will be patched by a command or script. This is executed through SSM and through Lambda scripts.
* These scripts will automate the flow of patching through the environments using the tags described.

Tags for Patch management:

|  |  |
| --- | --- |
| Key Name | Key Value |
| EnvironmentLevel | Production |
|  | Staging |
|  | QA |
|  | Development |

|  |  |
| --- | --- |
| Key Name | Key Value |
| ApplicationID | [custom] - (Found in CUSTOMER ServiceNow APM, custom value per project) |

Technical approach per Operating System

* **Windows Patching** 
  + WSUS repository and reporting
  + SSM command execution, for deployment of groups of systems
  + Scheduled Tasks for timed executions
  + CloudTrail for API logging
* **Linux Patching**
  + Switch to AWS AMI
  + Use AWS Package Repo
  + Updates will be done through the non-prod account and then pushed the Prod account.
  + CloudTrail for API logging

Additional technical approaches (AWS specific)

* **Amazon EC2** (e.g. Service Level 1, 2 and 3)
  + Develop a script to patch Windows and Linux instances using Simple Systems Manager (SSM)
    - One script to target each OS, or Specialized Tagged Application
    - Exclude EC2 instances that were launched from an auto scaling group
* **Amazon AMI Environment Roll** (This is Service Level 3 and 4. Potential any other system that cannot be auto scaled.)
  + Develop a script to patch Windows and Linux instances using Simple Systems Manager (SSM)
    - Each script will create an AMI of the current production system.
    - These AMI’s will be used to build a QA and Staging environment.
    - Once testing is complete, a new AMI will be created or the ELB will be switched between the production and staging environment.
* **Amazon EC2 - Auto Scaling** (This is for Service Level 3,4 and 5)
  + Develop a script to patch and create a new AMI to be assigned to a new launch configuration
    - Launch new EC2 based on latest launch configuration AMI
    - Patch using SSM
    - Clone current launch configuration and assign newly patched AMI
    - Assign new launch configuration to the existing auto scaling group
    - Force the launch of the newly patched EC2 instance

## Platform Logging

Description

There are two primary services available from AWS to aid in platform logging for your systems. These are Amazon CloudWatch and AWS CloudTrail.

* Amazon CloudWatch is a monitoring service for AWS cloud resources and the applications you run on AWS. You can use Amazon CloudWatch to collect and track metrics, collect and monitor log files, set alarms, and automatically react to changes in your AWS resources. You can also use CloudWatch logs to monitor and troubleshoot your systems and applications using your existing system, application, and custom log files. In addition, your existing system, application, and custom log files can be sent to CloudWatch Logs and monitored in near real-time.
* AWS CloudTrail provides a history of AWS API calls for your account, including API calls made via the AWS Management Console, AWS SDKs, command line tools, and higher-level AWS services (such as AWS CloudFormation). The AWS API call history produced by CloudTrail enables security analysis, resource change tracking, and compliance auditing.

Technical Approach

CUSTOMER’s current plan is to use the existing Splunk instance for historic log review and near real-time CloudTrail for event review; however, this will be evaluated on an application by application basis because using CloudWatch logs, fed by CloudTrail, you can automate responses to known issues very easily. This automated response to known issues may also be available with Splunk, but further investigation and testing are needed.

* VPC Flow logs will be used to review specific network traffic for security related tasks, system health and load review.
* Using the tagging described in this document, scripts will be developed that control the resiliency of the systems that are in the monitoring/logging systems
* Logging systems will notify, via SNS to the associated teams based on the tagging systems.
* AWS Config will be used to hold the system to specific in specific known configurations.
* SNS will send notification to the Administration team
* AWS Config will validate that patches and updates are in place and remediated correctly.
* (Optional) You can enable log file encryption, which provides an extra layer of security for your log files. For more information, see [Encrypting CloudTrail Log Files with AWS KMS–Managed Keys (SSE-KMS)](http://docs.aws.amazon.com/awscloudtrail/latest/userguide/encrypting-cloudtrail-log-files-with-aws-kms.html).

## Configuration Management

Using AWS Config

AWS Config is used to evaluate the configuration settings of your AWS resources. You do this by creating AWS Config rules, which represent your ideal configuration settings. AWS Config provides customizable, predefined rules to help you get started. You can also create your own custom rules from scratch. While AWS Config continuously tracks the configuration changes that occur among your resources, it checks whether these changes violate any of the conditions in your rules. If a resource does violate a rule, AWS Config flags the resource and the rule as noncompliant.

The AWS Config console shows you the compliance status of your rules and resources. You can use the console to assess how your AWS resources comply overall with your desired configurations, and you can learn which specific resources are noncompliant and which configuration attributes are the cause. You can also use the AWS CLI, the AWS Config API, and AWS SDKs to make requests to the AWS Config service for compliance information.

By using AWS Config to evaluate your resource configurations, you can more easily assess how well your resource configurations comply with internal practices, industry guidelines, and regulations.

This tool will allow CUSTOMER to validate tagging policies, resources, and security groups. This also allows for the coding of custom Lambda scripts to check for more specific items. In the case of the CUSTOMER Cloud Operations team, this will most likely include application of security groups, OS patches and other non-application specific items.

Example AWS Lambda Functions for AWS Config Rules (Node.js)

AWS Lambda executes functions in response to events that are published by AWS services. The function for a custom Config rule receives an event that is published by AWS Config, and the function then evaluates whether the configuration data provided by the event complies with the rule. The operations in a function for a Config rule differ depending on whether it performs an evaluation that is triggered by configuration changes or triggered periodically.

For information about common patterns within AWS Lambda functions, see [Programming Model](http://docs.aws.amazon.com/lambda/latest/dg/programming-model-v2.html) in the AWS Lambda Developer Guide. AWS Config will invoke a function like the following example when it detects a configuration change for a resource that is within a custom rule's scope.

If you use the AWS Config console to create a rule that is associated with a function like this example, choose Configuration changes as the trigger type. If you use the AWS Config API or AWS CLI to create the rule, set the MessageType attribute to ConfigurationItemChangeNotification. This example evaluates an Amazon EC2 instance and checks whether its instance type matches a specified value (for example, t2.micro).

Code Sample:

**var** aws = **require**('aws-sdk'), // Loads the AWS SDK for JavaScript.

config = **new** aws.ConfigService(), // Constructs a service object to use the aws.ConfigService class.

COMPLIANCE\_STATES = {

COMPLIANT: 'COMPLIANT',

NON\_COMPLIANT: 'NON\_COMPLIANT',

NOT\_APPLICABLE: 'NOT\_APPLICABLE'

};

// Receives the event and context from AWS Lambda. You can copy this handler and use it in your own

// code with little or no modification.

exports.handler = **function**(event, context, callback) {

// Parses the invokingEvent and ruleParameters values, which contain JSON objects passed as strings.

**var** invokingEvent = **JSON**.parse(event.invokingEvent),

ruleParameters = **JSON**.parse(event.ruleParameters),

compliance = COMPLIANCE\_STATES.NOT\_APPLICABLE,

putEvaluationsRequest;

**if** (isApplicable(invokingEvent.configurationItem, event)) {

compliance = evaluateCompliance(invokingEvent.configurationItem, ruleParameters);

}

// Initializes the request that contains the evaluation results.

putEvaluationsRequest = {

Evaluations: [

{

// Applies the evaluation result to the resource published in the event.

ComplianceResourceType: invokingEvent.configurationItem.resourceType,

ComplianceResourceId: invokingEvent.configurationItem.resourceId,

ComplianceType: compliance,

OrderingTimestamp: invokingEvent.configurationItem.configurationItemCaptureTime

}

],

ResultToken: event.resultToken

};

// Sends the evaluation results to AWS Config.

config.putEvaluations(putEvaluationsRequest, **function** (err, data) {

**if** (err) {

callback(err, null);

} **else** {

**if**(data.FailedEvaluations.length > 0) {

// Ends the function execution if any evaluation results are not successfully reported.

callback(null, **JSON**.stringify(data));

} **else** {

callback(null, data);

}

}

});

};

// Checks whether the resource has been deleted or is out of scope. If so, the evaluation is reported as

// 'NOT\_APPLICABLE'. You can copy this function and use it in your own code with little or no change.

**function** **isApplicable**(configurationItem, event) {

**var** status = configurationItem.configurationItemStatus,

eventLeftScope = event.eventLeftScope;

**return** (status === 'OK' || status === 'ResourceDiscovered') && eventLeftScope === false;

}

// Evaluates the resource and returns the compliance value to the handler.

**function** **evaluateCompliance**(configurationItem, ruleParameters) {

// Designates the resources as not applicable if it is not an EC2 instance.

**if**(configurationItem.resourceType !== 'AWS::EC2::Instance') {

**return** COMPLIANCE\_STATES.NOT\_APPLICABLE;

}

// Designates the resources as compliant if it is an EC2 instance of the desired type.

**if**(configurationItem.configuration.instanceType === ruleParameters.desiredInstanceType) {

**return** COMPLIANCE\_STATES.COMPLIANT;

}

**return** COMPLIANCE\_STATES.NON\_COMPLIANT;

}

# Provisioning & Service Catalog

## Provisioning

Description

To increase the identification, creation, and re-use of architecture patterns, all deployments (when possible and feasible) on AWS will be done through the use of CloudFormation scripting. AWS CloudFormation enables you to create and provision AWS infrastructure deployments predictably and repeatedly. It helps you leverage AWS products such as Amazon EC2, Amazon Elastic Block Store, Amazon SNS, Elastic Load Balancing, and Auto Scaling to build highly reliable, highly scalable, cost-effective applications in the cloud without worrying about creating and configuring the underlying AWS infrastructure. AWS CloudFormation enables you to use a template file to create and delete a collection of resources together as a single unit (a stack).

Technical Approach

AWS CloudFormation scripts will be used to deploy through Development, QA, Staging, and Production. In the future, ServiceNow could be used by the requester to fill out the required tags / parameters that would then be dynamically populated at launch time into AWS CloudFormation.

* CloudFormation templates are stored in AWS CodeCommit. Contact the Cloud Engineering team for access.
* Naming convention for templates - [To be determined by CUSTOMER during future iterations]

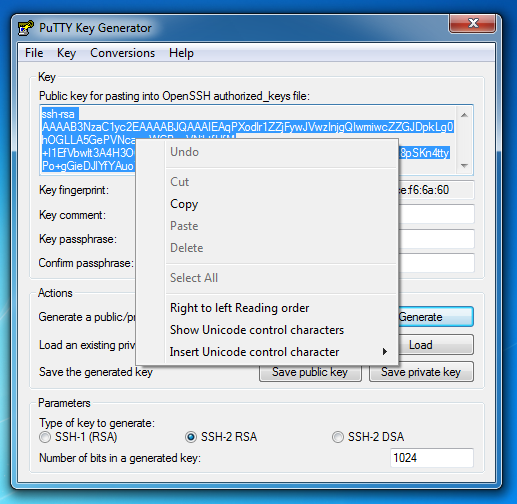
CloudFormation templates are handled in two files, the template file and the parameter file. The first, template, file is used to define the resources and flow of creation

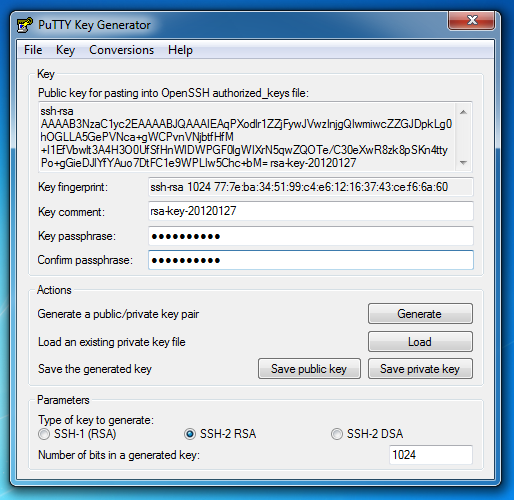
* S3 Bucket name with the templates that are going to be used.
* Under this bucket, a folder named “StackConfigs”
* Template:
  + https://s3.amazonaws.com/YourBucket/YourFolder/YourTemplate.template
* Parameter:
  + <https://s3.amazonaws.com/YourBucket/YourFolder/StackConfigs/YourParameters.json>

Code Repository (AWS CodeCommit)

* Each Repository in the CodeCommit should be limited to one Project
* Each BU should have multiple project in the repo
* Each project should include both the application code and the infrastructure code for that project and should be able to be deployed in parallel or by themselves.
* Each piece of a deployment should be broken up by the team responsible for that piece. i.e. Each layer should be deployable without impact to other layer of the application.

Setup for CodeCommit with the Team Chosen TortiseGit (<https://tortoisegit.org/)>

1. Setup a repo in the CodeCommit service, this should be a project based name.
2. Setup an IAM role for the project specifying the code CodeCommit repo
   1. 
   2. Change “Resource” : “\*” to “Resource” : “Your Repo ARN”
   3. Add any needed actions
3. Add a user with a policy or group that allows the usage needed for the application
4. TortoiseGIT comes with Puttygen, so let's use it to create your keys: Start > TortoiseGIT > Puttygen
5. Click Generate > (2) Copy the Public OpenSSH key > (3) Enter a passphrase (remember it - yes, you need a passphrase) > (4) Save the Private Key  
   Once the key is generated, you should copy it onto your clipboard. You will use this later to authenticate.
6. Afterwards, choose a memorable password and confirm it. Don't forget your password!
7. Finally, click on the "Save private key" button and save your private key somewhere you'll remember.
8. Copy the public key from the window into the public ssh key section of the IAM user being setup
9. Copy the KEYID that is created from this.
10. Open a the Tortoise dialog and select configure
11. Go to "TortoiseGit > Settings > Credential"
12. Click the “G” and add:  
    !aws codecommit credential-helper $@  
    To the helper Line and the KEYID to the Username. Add the Repo to the URL, but only include the domain name to the .com (<https://git-codecommit.us-east-1.amazonaws.com>)
13. You should now be able to clone a CodeCommit Repo.



## Service Catalog

Description

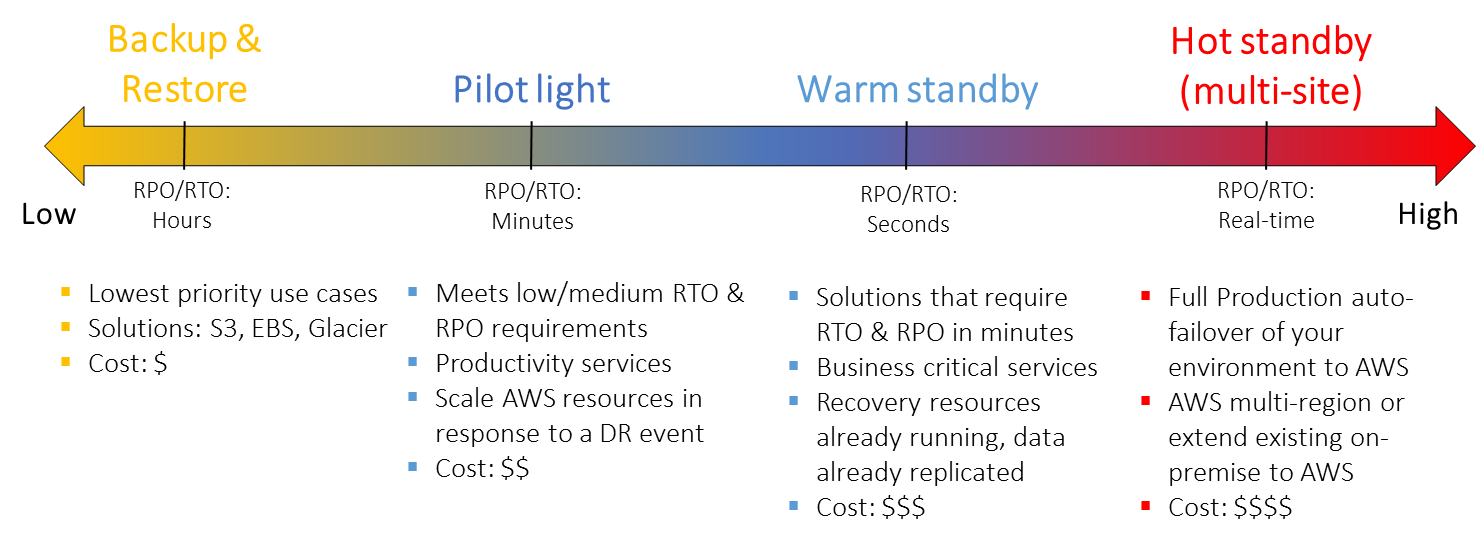
In the future, a service catalog (e.g. Service Catalog) could be used by the requester to fill out the required tags / parameters that would then be dynamically populated at launch time into CloudFormation.

Technical Approach

[To be determined by CUSTOMER during future iterations]

# Availability Management

Availability Management encompasses multiple aspects of designing for and operating to insure the availability of applications and services to support the business service level agreement. This includes restoring applications based on Recovery Point and Time objectives. Recovery Point Objectives (RPO) defines how much data loss can be tolerated. While Recovery Time Objective (RTO) defines how quickly the service needs to be restored. Each application should be evaluated based on the business criticality to determine the RTO/RPO requirements. Most organizations classify applications using tiers to convey which recovery option is appropriate for the given application.



Through the use of High Availability (keep your solution running), Backup (make sure your data is safe), and Disaster Recovery (get your data back after a major disaster) the cloud provides comprehensive strategies to meet the desired user experience, SLO/SLA, risk, and cost objectives for each of your applications.

## Backup & Restore

Description

The Backup and Restore approach to availability management gives low priority applications a reasonably easy to set-up and cost-effective method to restore an application’s availability to its users and to protect from data loss.

Key tenets of backup and restore:

1. Select an appropriate tool or method to back up your data into AWS.

2. Ensure that you have an appropriate retention policy for this data.

3. Ensure that appropriate security measures are in place for this data, including encryption and access policies.

4. Regularly test the recovery of this data and the restoration of your system

Technical Approach

|  |  |
| --- | --- |
| Key Name | Key Value [pattern] |
| BackupPolicy | [schedule]:[retention] |

This Key Value above means:

* [schedule]: A schedule Key Value of 2330|30|7 means the application is snapshotted at 11:30pm UTC, every day of the month, every day of the week
* [retention]: A retention Key Value of 1|7|4|6|2|2 means the snapshots will be retained on the following schedule: 1 backup a day stored (important to set if you do more than one backup a day), 7 daily backups stored, 4 weekly backups stored, 6 monthly backups stored, 2 semi-annual (every 6 months) backups stored, 2 yearly backups stored

## Pilot Light

Description

The term pilot light is often used to describe a disaster recovery scenario in which a minimal version of an environment is always running in the cloud. The idea of the pilot light is an analogy that comes from the gas heater. In a gas heater, a small flame that’s always on can quickly ignite the entire furnace to heat up a house. This scenario is similar to a backup-and-restore scenario. For example, with AWS you can maintain a pilot light by configuring and running the most critical core elements of your system in AWS. When the time comes for recovery, you can rapidly provision a full-scale production environment around the critical core.

Infrastructure elements for the pilot light itself typically include your database servers, which would be replicating data to Amazon EC2. Depending on the system, there may be other critical data outside of the database that needs to be replicated to AWS. This is the critical core of the system (the pilot light) around which all other infrastructure pieces in AWS can quickly be provisioned (the rest of the furnace) to restore the complete system.

Key tenets of Pilot Light:

1. Select an appropriate tool or method to back up your data into AWS.

2. Ensure that you have an appropriate retention policy for this data.

3. Ensure that appropriate security measures are in place for this data, including encryption and access policies.

4. Regularly test the recovery and the restoration of your system

Technical Approach

To provision the remainder of the infrastructure to restore business critical services, there will be some pre-configured servers bundled as Amazon Machine Images (AMIs), which are ready to be started up at a moment’s notice. When starting recovery, instances from these AMIs come up quickly and find their role within the deployment around the pilot light.

From a networking point of view, Elastic Load Balancing will be used to distribute traffic to multiple instances. The DNS records will need to be updated to point at your Amazon EC2 instance or point to your Elastic Load Balancing using a CNAME.

Key points for preparation:

* Set up EC2 instances to replicate or mirror data.
* Ensure that you have all supporting custom software packages available in AWS.
* Create and maintain Amazon Machine Images (AMI) of key servers where fast recovery is required.
* Regularly run these servers, test them, and apply any software updates and configuration changes.
* Create AWS CloudFormation templates for the provisioning of the AWS resources

Based upon discussion with CUSTOMER – this approach will be utilized for Service Level 3 and Recovery Tier 3 and below.

## Warm Standby

Description

The term warm standby is used to describe a DR scenario in which a scaled-down version of a fully functional environment is always running in the cloud. A warm standby solution extends the pilot light elements and preparation. It further decreases the recovery time because some services are always running. By identifying your business-critical systems, you can fully duplicate these systems on AWS and have them always on.

Key tenets of Warm Standby:

1. Select an appropriate tool or method to back up your data into AWS.

2. Ensure that you have an appropriate retention policy for this data.

3. Ensure that appropriate security measures are in place for this data, including encryption and access policies.

4. Regularly test the recovery and the restoration of your system

Technical Approach

These servers can be running on a minimum-sized fleet of Amazon EC2 instances on the smallest sizes possible. This solution is not scaled to take a full-production load, but it is fully functional. It can be used for non-production work, such as testing, quality assurance, and internal use.

Key points for preparation: Amazon Web Services

* Set up EC2 instances to replicate or mirror data.
* Create and maintain Amazon Machine Images (AMIs).
* Run your application using a minimal footprint of EC2 instances or AWS infrastructure.
* Patch and update software and configuration files in line with your live environment and as per the Patch Process outlined in this document.

Based upon discussion with CUSTOMER – this approach could be utilized for Service Level 3 and 4 as well as Recovery Tier 2 or 3.

## Multi-site (AZ)

Description

A multi-site solution runs in AWS as well as on your existing on-site infrastructure, in an active-active configuration. The data replication method that you employ will be determined by the recovery point that you choose. In addition to recovery point options, there are various replication methods, such as synchronous and asynchronous methods.

A weighted DNS service, such as Amazon Route 53, is used to route production traffic to the different sites. A proportion of traffic will go to your infrastructure in AWS, and the remainder will go to your on-site infrastructure. In an on-site disaster situation, you can adjust the DNS weighting and send all traffic to the AWS servers. The capacity of the AWS service can be rapidly increased to handle the full production load. EC2 Auto Scaling can be used to automate this process. You may need some application logic to detect the failure of the primary database services and cut over to the parallel database services running in AWS.

The cost of this scenario is determined by how much production traffic is handled by AWS in normal operation. In the recovery phase, you only pay for what you use in addition and for the duration that the DR environment is required at full scale. You can further reduce cost by purchasing Reserved Instances for your “always on” AWS servers.

Key tenets of Multi-site (AZ):

1. Select an appropriate tool or method to back up your data into AWS.

2. Ensure that you have an appropriate retention policy for this data.

3. Ensure that appropriate security measures are in place for this data, including encryption and access policies.

4. Regularly test the recovery and the restoration of your system

Technical Approach

Scenario: A site with regional needs. This application is built in two AZ’s which can answer for each other and are both active at all times.

* Set up your AWS environment to duplicate your production environment.
* Set up DNS weighting or similar technology to distribute incoming requests to both sites.
* This can be the most complicated implementation – it does require more effort and some development work, specifically for applications that require sticky sessions.

Based upon discussion with CUSTOMER – this approach could be utilized for Service Level 4 and 5 as well as Recovery Tier 1 or 2.

## Other Availability Considerations

As an additional consideration, regardless of which pattern is used, to aid in application availability, it is recommended to create and adjust applications to support the use of multiple AZ’s in one region. By deploying a system that is spread between two different AZ’s, the use of ELB’s to split the traffic being sent to the system allows for a complete failure of one AZ.

This does require the application to be able to be split into two AZ’s. At very least this can be used to create two active sites that are coupled and a client will be aware of a very short outage during a failure. Ideally these applications are less coupled and able to support applications that shift from one AZ to another.

This approach is suitable for Service Level 3,4,5 and Recovery Tier 1 and 2

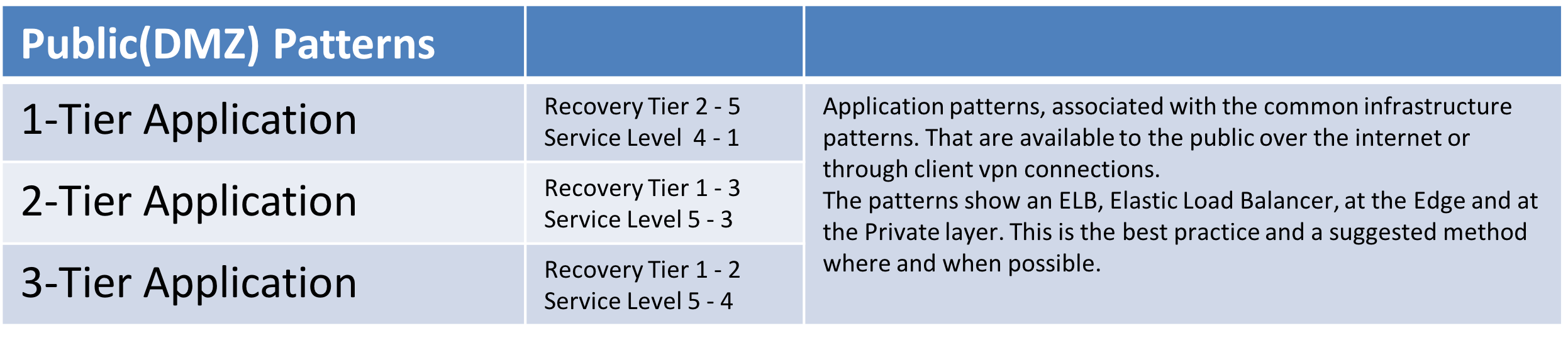
# Platform Lifecycle Management

CUSTOMER has chosen to create, use and enforce standard architecture practices wherever possible to decrease one-off solutions. This increases the simplicity of training Cloud Operations staff. The details, including graphical representations of these architectures are included below. The scripts defined here are being written current by CUSTOMER, based on scripts that they have been given.

## Application Architecture Patterns:

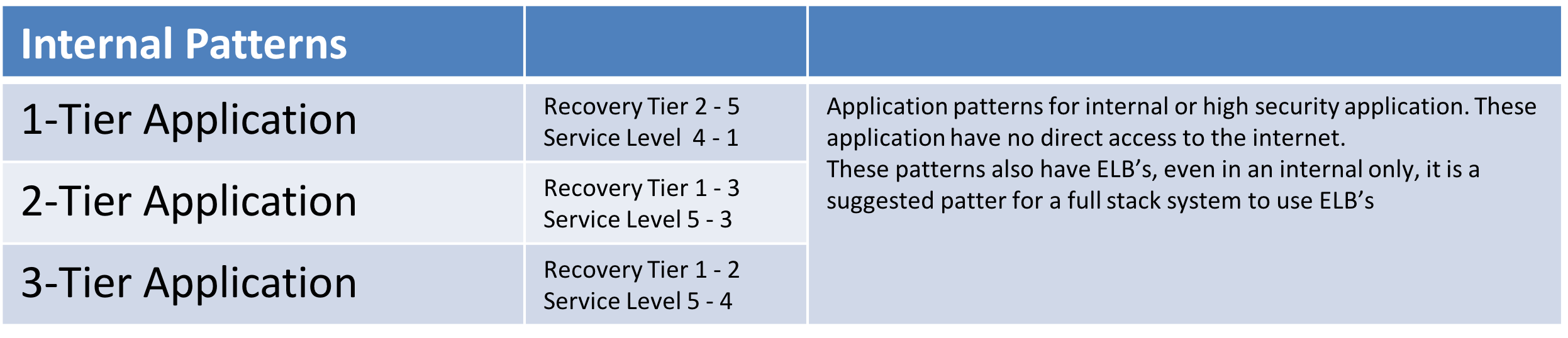
Public (DMZ) Patterns

Application patterns, associated with the common infrastructure patterns. These are available to the public over the internet or through client VPN connections. The patterns show an ELB, Elastic Load Balancer, at the Edge and at the Private layer. This is the best practice and a suggested method where and when possible.



Internal Patterns

Application patterns for internal or high security application. These application have no direct access to the internet. These patterns also have ELB’s, even in an internal only, it is a suggested patter for a full stack system to use ELB’s.



Public (DMZ) Patterns

|  |  |
| --- | --- |
| DMZ 1-Tier Application Architecture |  |
| Template: **DMZ1TIER.template**  Included 1 instance per AZ in a public tier.  Parameters: DMZ1TIER.json  Launcher: AWSLauncher.sh  Command line:  **AWSLauncher.sh StackName Region S3BucketName**  (./AWSLauncher.sh test10 us-east-1 s3.amazon.com/Bucketname)  Suitable:  Recovery Tier 2 - 5  Service Level 4 - 1 |  |

|  |  |
| --- | --- |
| DMZ 2-Tier Application Architecture |  |
| Template: **DMZ2TIER.template**  Included 1 server per AZ and an RDS solution in the Private tier.  Parameters: **DMZ2TIER.json**  Launcher: **AWSLauncher.sh**  Command line:  **AWSLauncher.sh StackName Region S3BucketName**  (./AWSLauncher.sh test10 us-east-1 s3.amazon.com/Bucketname)  Suitable:  Recovery Tier 1 - 3  Service Level 5 - 3 |  |

|  |  |
| --- | --- |
| DMZ 3-Tier Application Architecture |  |
| Template: **DMZ3TIER.template**  Included 1 server per Private and Public tier and AZ and an RDS solution in the Private tier.  Parameters: **DMZ3TIER.json**  Launcher: **AWSLauncher.sh**  Command line:  **AWSLauncher.sh StackName Region S3BucketName**  (./AWSLauncher.sh test10 us-east-1 s3.amazon.com/Bucketname)  Suitable:  Recovery Tier 1 - 2  Service Level 5 - 4 |  |

Internal Patterns

|  |  |
| --- | --- |
| Internal 1-Tier Application |  |
| Template: **Internal1TIER.template**  Included 1 server per AZ.  Parameters: **Internal1TIER.json**  Launcher: **AWSLauncher.sh**  Command line:  **AWSLauncher.sh StackName Region S3BucketName**  (./AWSLauncher.sh test10 us-east-1 s3.amazon.com/Bucketname)  Suitable:  Recovery Tier 1 - 3  Service Level 5 - 3 |  |

|  |  |
| --- | --- |
| Internal 2-Tier Application |  |
| Template: **Internal1TIER.template**  Included 1 server per AZ.  Parameters: **Internal1TIER.json**  Launcher: **AWSLauncher.sh**  Command line:  **AWSLauncher.sh StackName Region S3BucketName**  (./AWSLauncher.sh test10 us-east-1 s3.amazon.com/Bucketname)  Suitable:  Recovery Tier 1 - 2  Service Level 5 - 4 |  |

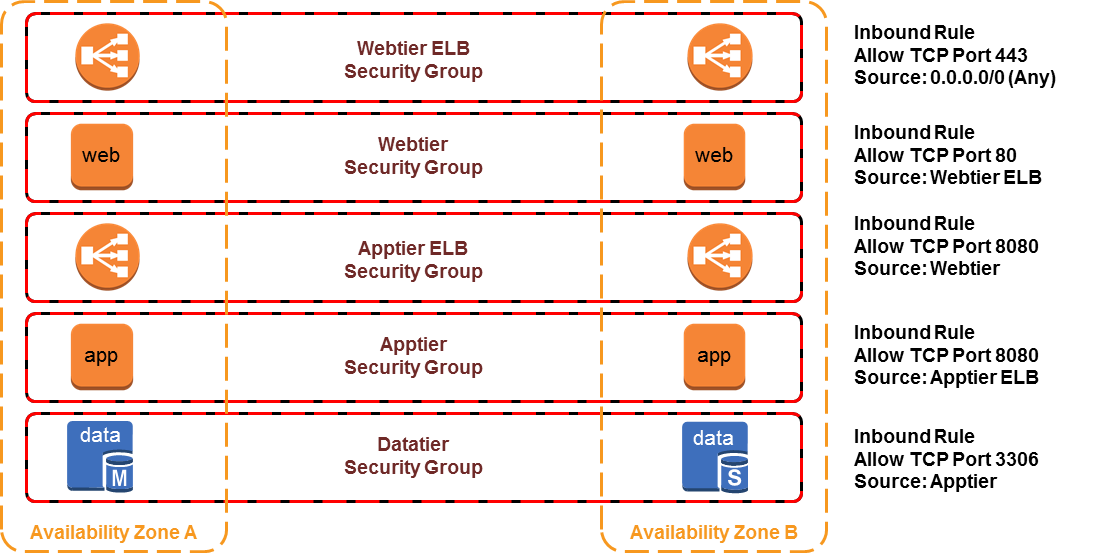
|  |  |
| --- | --- |
| Internal 3-Tier Application |  |
| Template: **Internal2TIER.template**  Included 1 server per AZ and an RDS solution in the Private tier.  Parameters: **Internal2TIER.json**  Launcher: **AWSLauncher.sh**  Command line:  **AWSLauncher.sh StackName Region S3BucketName**  (./AWSLauncher.sh test10 us-east-1 s3.amazon.com/Bucketname)  Suitable:  Recovery Tier 1 - 2  Service Level 5 - 4 |  |

## Security Architecture Patterns

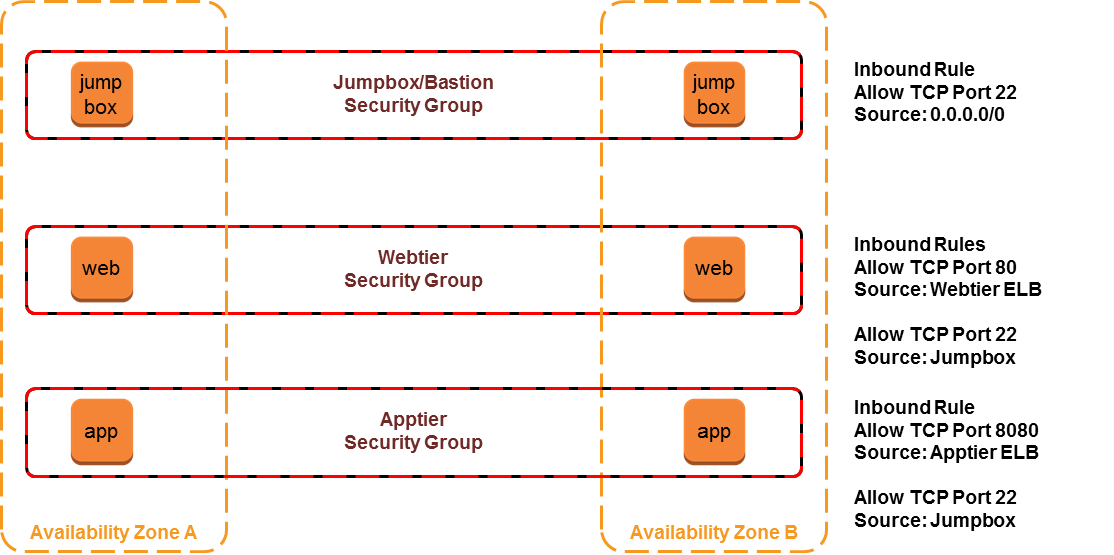
A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. When you launch an instance in a VPC, you can assign the instance to up to five security groups. Security groups act at the instance level, not the subnet level. Therefore, each instance in a subnet in your VPC could be assigned to a different set of security groups. If you don't specify a particular group at launch time, the instance is automatically assigned to the default security group for the VPC.

For each security group, you add rules that control the inbound traffic to instances, and a separate set of rules that control the outbound traffic. This section describes the basics things you need to know about security groups for your VPC and their rules.

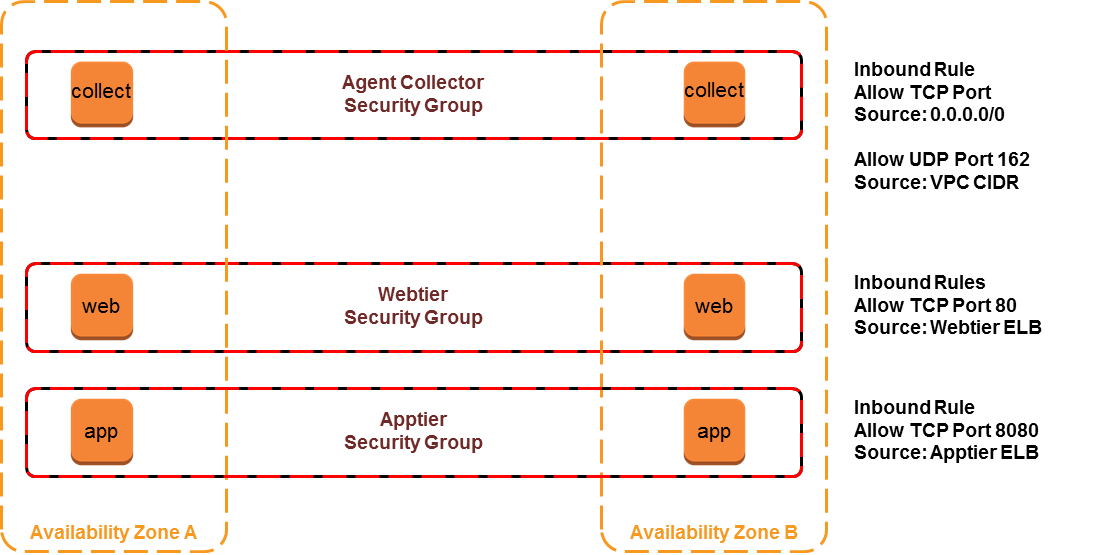
Security Group Architecture Per Application Tier



Security Group Architecture One-to-Many



Security Group Architecture Many-to-One



## AMI Lifecycle Management

One of the key benefits of cloud is a highly automated process for patch management. Traditionally organizations have struggled to roll out patches due to enormous time and complexity around patch roll-out. Treating Operating system as a code and automating the process around build and deploy is a trait of an organization extracting value out of cloud.

Amazon Web Services (AWS) offers its customers several methods that make it easier to provision Amazon Elastic Compute Cloud (Amazon EC2) instances and store instance configurations across a variety of different server and application deployment models. The most common unit of management is the Amazon Machine Image (AMI), which provides the information required to launch an EC2 instance. AWS customers specify an AMI when launching an instance and can use a single AMI to launch multiple EC2 instances.

General Best Practices:

All organizations should have a documented process for provisioning server images to ensure images can be recreated and easily updated in adherence to corporate standards. When determining what to include in an AMI, consider the following AMI design best practices:

* **Security:** Avoid embedding passwords, private keys, or other sensitive information in AMIs. Configuration is idempotent**,** manual changes are detected and rolled back
* **Automation**: Leverage AWS CloudFormation or a third-party configuration management tool to document and automate AMI creation and updating
* **Infrastructure as Code**: Create a library of reusable, modular templates that can be programmatically assembled to create different types of AMIs
* **Monitor**: Instrument AMIs with a standard bootstrapping capability that allows the instance to reference runtime information at launch.
* **Governance**: Develop a consistent strategy for tagging AMIs to allow for the easy organization and identification of images and their contents
* **Process**: Develop a patching strategy aligned with corporate security and application lifecycle. Ability to continuously push newer AMI’s and enforcing applications to use the newer AMI’s is important. Design for all types of use cases like (OS patching, auto-scale event, bad instance, upgrading instance type, critical security patch)

Sources of AMI:

* “Golden” AMIs created by customer
* Amazon-maintained AMIs
* Public AMIs from other organizations
* AMIs generated from imported virtual machines
* AWS Marketplace AMIs

AMI Design Options:

There are two primary approaches to design an AMI:

* **Fully Baked AMI:** purpose-built to deploy a complete running instance, including the installation and configuration of all required software
* **Just enough Operating System (JeOS):** include only minimal configurations and software before then dynamically installing the required packages on first boot

Factors to consider:

* How quickly do you need to be able to recover a failing instance or add additional compute capacity?
* Does the workload’s baseline stay static for a relatively long period?
* Does your server configuration require manual provisioning or configuration?
* Do you need to minimize the complexity of deploying resources to both AWS and on-premises environments?
* Is there existing server provisioning tools or processes that you are trying to align with AWS?

Examples:

* **Standard Apache WebServer**: Will have a standard AMI with apache installed and completely hardened, on boot AMI will pick up application specific configuration and assets from repository like S3
* **Custom Application with long install process**: Fully baked AMI specific for each environment will have to be maintained if the application deployment or configuration process cannot be automated or takes a long time

AMI Management Options:

* **“Patch-in-Place”:** leverage current or new customer owned patching infrastructure to patch instances
  + deploy all at once
  + rolling update
* **“Replace and Retire”:** build a new environment side by side with old environment, gradually provision load to new environment, retire old one.

Patching Considerations:

* Make a decision on which AMIs you will leverage
  + Do they need to be part of your domain?
  + Do they need to be tracked by your asset management tool?
  + Do they need to be scanned by the security team on a regular basis?
  + How will a reboot of a server affect your service availability?
* What is the security team’s policy for patching domain servers?
* Are the edge cases that must not be patched because of OS and Application functionality?
* Use the configuration management tool of your choice to layer everything else on top of your AMIs
* AMIs are region bound, as such patching needs to happen in every region.

View the [AMI Design](https://d0.awsstatic.com/aws-answers/AWS_AMI_Design.pdf) Whitepaper for more details:

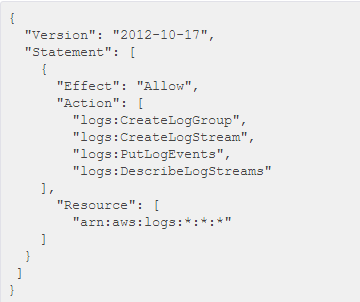
https://d0.awsstatic.com/aws-answers/AWS\_AMI\_Design.pdf

# Event Monitoring

To aid in the monitoring of the appropriate environments (ie. Prod/QA/PreProd0, CUSTOMER will use Amazon CloudWatch to collect and track metrics, collect and monitor log files, set alarms, and enable the ability to automatically react to changes in the AWS resources. Amazon CloudWatch can be used to monitor AWS resources such as Amazon EC2 instances, Amazon DynamoDB tables, and Amazon RDS DB instances, as well as custom metrics generated by your applications and services, and any log files your applications generate. The use of Amazon CloudWatch will provide the CUSTOMER Cloud Operations team with the ability to gain system-wide visibility into resource utilization, application performance, and operational health.

Installing the Cloudwatch Agent:

* **To configure your IAM role or user for CloudWatch Logs**
* Open the Identity and Access Management (IAM) console at <https://console.aws.amazon.com/iam/>.
* In the navigation pane, click **Roles**, and then in the **Role Name** column, click an IAM role.
* On the **Permissions** tab, under **Inline Policies**, click **Create Role Policy**.
* On the **Set Permissions** page, click **Custom Policy**, and then click **Select**.
* For more information about creating custom policies, see [IAM Policies for Amazon EC2](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-policies-for-amazon-ec2.html)in the *Amazon EC2 User Guide for Linux Instances*.
* On the **Review Policy** page, in the **Policy Name** field, type a name for the policy.
* In the **Policy Document** field, paste in the following policy:



Dashboarding with AWS Cloudwatch

Dashboards offer direct views into instance health and resource usage. The usage of the described tags:

* ServiceLevel
* RecoveryTier
* EnvironmentLevel
* ApplicationID
* CostCenter

# License Management

CUSTOMER will utilize their current License management system. Internal decisions and workflow to be created with asset management group.

# Network Architecture

CUSTOMER’s VPC and Subnet patterns:

* One account for Production
* One account for Non-production
* One account for POC
  + This separates the code and network functionality so that accidental pushes to production are reduced.

During the AWS Platform Jumpstart engagement with AWS ProServe, CUSTOMER decided on the following Network, VPC, AZ, and Connectivity decisions:

* Approved Regions
  + US-EAST- 1 (Virginia)
  + US-WEST-2 (Oregon)
  + EU-WEST (Ireland)
  + AP-NORTHEAST-1 (Tokyo)
* Number of Availability Zones
  + Minimum and maximum of 2 Availability Zones per Region
* Number of VPC’s
  + 3 Amazon VPCs in Prod AWS account (Prod, Management and Workspaces)
  + 3 Amazon VPC in Non-Prod AWS account (Prod, Management and Workspaces)
  + 1 Amazon VPC in Lab/POC AWS account
  + Any number of Amazon VPCs in Fuse AWS account
  + Any number of Amazon VPCs for Pivotal Cloud Foundry
  + 4 Amazon VPCs for Dev, Stage, QA, and Prod in CustomerHealth @ Home AWS account
* VPC Connectivity
  + Management VPC will be peered to all VPCs in the same region
  + VPN connections initially, being the process of instantiating Direct Connect lines
    - Establish 4 AWS Direct Connect lines in 4 AWS Regions
      * Dublin Ohio Data Center
        + 1 - 10Gbs line and port speed to US-EAST-1
        + 1 - 10Gbs line and port speed to US-WEST-2
        + 1 - 30Mbs line and 1 Gbs port speed to EU-WEST
        + 1 - 30Mbs line and 1 Gbs port speed to AP-NORTHEAST-1
      * McGaw Illinois Data Center
        + 1 - 10Gbs line and port speed to US-EAST-1
        + 1 - 10Gbs line and port speed to US-WEST-2
        + 1 - 30Mbs line and 1 Gbs port speed to EU-WEST
        + 1 - 30Mbs line and 1 Gbs port speed to AP-NORTHEAST-1

## CloudFormation patterns

* All resources should be scripted using AWS CloudFormation
* Each layer of resources is built as an isolated stack
  + One Subnet for Edge per AZ which his used for a url filter, logging system and other HIPAA compliance needs.
  + One Public Subnet net per AZ for public facing systems. These systems will default run through the edge subnet and through the edge security systems if possible
  + One Private Subnet per AZ that is directly connect to the VPN and is only accessible from the core and the public subnet.
    - There will be exception to this security rule, but they should be documented.
  + These choices are driven by the Service Level that the application or portion of the Application has been assigned.

# Security

Refer to the Security Playbook / Runbook located at: [link]

(This engagement is currently underway with CUSTOMER and is scheduled for completion in July 2016. Once complete, this section should be completed either by integrating the Security Playbook into this document or referring to it by link to its location.)

# Financial Management

Trusted Advisor

Review AWS Trusted Advisor at least weekly to receive best practices (or checks) in four categories:

* Cost optimization (server utilization, recommend reserved instances, etc.)
* security
* fault tolerance
* performance improvement

The status of the check is shown by using color coding on the dashboard page: Red (take action), Yellow (review), Green (no action necessary).

Reserved Instances

* Amazon EC2 Reserved Instances allow you to reserve Amazon EC2 computing capacity for 1 or 3 years, in exchange for a significant discount (up to 75%) compared to On-Demand instance pricing.
* Reserved Instances can significantly lower your computing costs for your workloads and provide a capacity reservation so that you can have confidence in your ability to launch the number of instances you have reserved when you need them.

Further Considerations

* Turn Off Environments, when you are not using them
* Turn off Dev, QA and Staging environments when not needed
* Shutdown POC’s once finished
* Use data lifecycle policies to delete unneeded snapshots and data

Cost Reduction Opportunities on AWS

* Like-for-Like, Forklift to AWS
* Right-sizing instances on utilization [thin-provisioned on-premise = over-provisioned on AWS]
* Architecture Optimization using Cloud-native services
* Automation of known events, auto-solving issues
* Purchase AWS Reserved Instances
* AWS Enterprise Discount Program (EDP)
* n+1. Leverage future AWS innovative releases

Charge / Show back reports

Currently untagged items are being billed to a general bucket and should be reported. CUSTOMER is currently looking to have these machine shutdown after a period of time and sufficient warning have been delivered. This will allow the owner to step up and claim the instance in question. Currently a lambda function is being written to list the resources in an account and show a report split out by BU, CostCenter, ApplicationID and environment level. This should allow the cloud team the needed help in rounding up the lagging systems and showing the CUSTOMER internal clients how to lower their bills. These types of reports are dependent on tagging, so it’s importance is increased and the effort involved in correcting and enforcing the policy is key.