AWS Secure Environment Accelerator

Amazon Web Services

Amazon Web Services

None

Table of contents

1. Documentation

1.1

- Link to Accelerator releases and change history
- Sample configuration files and customization details
- State Machine behavior and inputs
- Chart containing details as to WHAT we do and WHERE we support it (regions, accounts, etc.)
- Accelerator central logging bucket structures
- Unofficial Accelerator Roadmap (GitHub projects) - *Please upvote desired features*

1.2

1.3

1.4

- Accelerator release process (AWS Internal)

1.5

1.6 (Early Draft)

- Accelerator Prescriptive Architecture Sample Diagrams

1.7

1.8

1.9

2. 1. Accelerator Basic Operation and Frequently asked Questions

2.1 1.1. Operational Activities

2.1.1 1.1.1. How do I add new AWS accounts to my AWS Organization?

- We offer four options and all can be used in the same Accelerator deployment. All options work with AWS Control Tower, ensuring the account is both ingested into Control Tower and all Accelerator guardrails are automatically applied:
- Users can simply add the following five lines to the configuration file `workload-account-configs` section and rerun the state machine. The majority of the account configuration will be picked up from the ou the AWS account has been assigned. You can also add additional account specific configuration, or override items like the default ou budget with an account specific budget. This mechanism is often used by customers that wish to programmatically create AWS accounts using the Accelerator and allows for adding many new accounts at one time.

```
json
  "fun-acct": {
    "account-name": "TheFunAccount",
    "email": "myemail+aseaT-funacct@example.com",
    "src-filename": "config.json",
    "ou": "Sandbox"
  }
```

1. We've heard consistent feedback that our customers wish to use native AWS services and do not want to do things differently once security controls, guardrails, or accelerators are applied to their environment. In this regard, simply create your new AWS account in AWS Organizations as you did before**, either by a) using the AWS Console or b) by using standard AWS account creation API's, CLI or 3rd party tools like Terraform.

- ** **IMPORTANT:** When creating the new AWS account using AWS Organizations, you need to specify the role name provided in the Accelerator configuration file `global-options\organization-admin-role`, otherwise we cannot bootstrap the account. In Control Tower installations, this **MUST** be set to `AWSControlTowerExecution`, for customers who installed prior to v1.2.5 this value is `AWSCloudFormationStackSetExecutionRole` and after v1.2.5 we were recommending using the role `OrganizationAccountAccessRole` as this role is used by default by AWS Organizations if no role name is specified when creating AWS accounts through the AWS console or cli.
- On account creation we will apply a quarantine SCP which prevents the account from being used by anyone until the Accelerator has applied the appropriate guardrails
- Moving the account into the appropriate OU triggers the state machine and the application of the guardrails to the account, once complete, we will remove the quarantine SCP.
- **NOTE:** Accounts CANNOT be moved between OU's to maintain compliance, so select the proper top-level OU with care
- In AWS Organizations, select ALL the newly created AWS accounts and move them all (preferably at once) to the correct destination OU (assuming the same OU for all accounts)
- In case you need to move accounts to multiple OU's we have added a 2 minute delay before triggering the State Machine
- Any accounts moved after the 2 minute window will NOT be properly ingested, and will need to be ingested on a subsequent State Machine Execution.

2. Create your account using Account Factory in the AWS Control Tower console.

No matter the mechanism you choose, new accounts will automatically be blocked from use until fully guardrailed, the Accelerator will automatically execute, and accounts will automatically be ingested into AWS Control Tower.

2.1.2 1.1.2. I tried to enroll a new account via Control Tower but it failed? The state machine failed during the "Load Organization Configuration" step with the error "The Control Tower account: ACCOUNT_NAME is in a failed state ERROR"?

If account enrollment fails within Control Tower, you will need to follow the troubleshooting steps here. A common reason for this is not having the `ControlTowerExectution` role created in the account you are trying to enroll. Even after you successfully enroll the account, it is possible the state machine will fail at `Load Organization Configuration`. If you look at the cloudwatch logs you will see the error message:

```
There were errors while loading the configuration: The Control Tower account: ACCOUNT_NAME is in a failed state ERROR.
```

This is because the Accelerator checks that there are no errors with Control Tower before continuing. In some cases Control Tower can leave an orphaned Service Catalog product in an **Error** state. You need to cleanup Control Towers Service Catalogs Provisioned Products so there are no products remaining in an error or tainted state before you can successfully re-run the state machine.

2.1.3 1.1.3. Can I use AWS Organizations for all tasks I currently use AWS Organizations for?

- In AWS Organizations you can continue to:
- create and rename AWS accounts
- move AWS accounts between ou's
- create, delete and rename ou's, including support for nested ou's
- create, rename, modify, apply and remove SCP's
- What can't I do:
- modify Accelerator or Control Tower controlled SCP's
- add/remove SCP's on top-level OU's (these are Accelerator and/or Control Tower controlled)
- users can change SCP's on non-top-level ou's and non-Accelerator controlled accounts as they please
- add/remove SCP's on specific accounts that have Accelerator controlled SCPs
- move an AWS account between top-level ou's (i.e. `Sandbox` to `Prod` is a security violation)
- moving between `Prod/sub-ou-1` to `Prod/sub-ou2` or `Prod/sub-ou2/sub-ou2a/sub-ou2ab` is fully supported
- create a top-level ou (need to validate, as they require config file entries)
- remove quarantine SCP from newly created accounts
- we do not support forward slashes (`/`) in ou names, even though the AWS platform does
- More details:
- If an AWS account is renamed, an account email is changed, or an OU is renamed, on the next state machine execution, the config file will automatically be updated.
- If you edit an Accelerator controlled SCP through Organizations, we will reset it per what is defined in the Accelerator configuration files.
- If you add/remove an SCP from a top-level ou or Accelerator controlled account, we will put them back as defined in the Accelerator configuration file.
- If you move an account between top-level ou's, we will put it back to its original designated top-level ou.
- The Accelerator fully supports nested ou's, customers can create any depth ou structure in AWS Organizations and add/remove/change SCP's *below* the top-level as they desire or move accounts between these ou's without restriction. Users can create ou's to the full AWS ou structure/ depth
- Except for the Quarantine SCP applied to specific accounts, we do not 'control' SCP's below the top level, customers can add/create/customize SCP's
- as of v1.3.3 customers can optionally control account level SCP's through the configuration file

2.1.4 1.1.4. How do I make changes to items I defined in the Accelerator configuration file during installation?

Simply update your configuration file in CodeCommit and rerun the state machine! In most cases, it is that simple.

If you ask the Accelerator to do something that is not supported by the AWS platform, the state machine will fail, so it needs to be a supported capability. For example, the platform does not allow you to change the CIDR block on a VPC, but you can accomplish this as you would today by using the Accelerator to deploy a new second VPC, manually migrating workloads, and then removing the deprecated VPC from the Accelerator configuration.

Below we have also documented additional considerations when creating or updating the configuration file.

It should be noted that we have added code to the Accelerator to block customers from making many 'breaking' or impactful changes to their configuration files. If someone is positive they want to make these changes, we also provide override switches to allow these changes to be attempted forcefully.

2.1.5 1.1.5. Can I update the config file while the State Machine is running? When will those changes be applied?

Yes. The state machine captures a consistent input state of the requested configuration when it starts. The running Accelerator instance does not see or consider any configuration changes that occur after it has started. All configuration changes occurring after the state machine is running will only be leveraged on the *next* state machine execution.

2.1.6 1.1.6. What if I really mess up the configuration file?

The Accelerator is designed with checks to compare your current configuration file with the version of the config file from the previous successful execution of the state machine. If we believe you are making major or breaking changes to the config file, we will purposefully fail the state machine. See 1.4. Config file and Deployment Protections for more details.

With the release of v1.3.0 we introduced state machine scoping capabilities to further protect customers, detailed here

2.1.7 1.1.7. What if my State Machine fails? Why? Previous solutions had complex recovery processes, what's involved?

If your main state machine fails, review the error(s), resolve the problem and simply re-run the state machine. We've put a huge focus on ensuring the solution is idempotent and to ensure recovery is a smooth and easy process.

Ensuring the integrity of deployed guardrails is critical in operating and maintaining an environment hosting protected data. Based on customer feedback and security best practices, we purposely fail the state machine if we cannot successfully deploy guardrails.

Additionally, with millions of active customers each supporting different and diverse use cases and with the rapid rate of evolution of the AWS platform, sometimes we will encounter unexpected circumstances and the state machine might fail.

We've spent a lot of time over the course of the Accelerator development process ensuring the solution can roll forward, roll backward, be stopped, restarted, and rerun without issues. A huge focus was placed on dealing with and writing custom code to manage and deal with non-idempotent resources (like S3 buckets, log groups, KMS keys, etc.). We've spent a lot of time ensuring that any failed artifacts are automatically cleaned up and don't cause subsequent executions to fail. We've put a strong focus on ensuring you do not need to go into your various AWS sub-accounts and manually remove or cleanup resources or deployment failures. We've also tried to provide usable error messages that are easy to understand and troubleshoot. As new scenario's are brought to our attention, we continue to adjust the codebase to better handle these situations.

Will your state machine fail at some point in time, likely. Will you be able to easily recover and move forward without extensive time and effort, YES!

2.1.8 1.1.8. How do I update some of the supplied sample configuration items found in reference-artifact, like SCPs and IAM policies?

To override items like SCP's or IAM policies, customers simply need to provide the identically named file in there input bucket. As long as the file exists in the correct folder in the customers input bucket, the Accelerator will use the customers supplied version of the configuration item, rather than the Accelerator version. Customer SCP's need to be placed into a folder named `scp` and iam policies in a folder named `iam-policy` (case sensitive).

The Accelerator was designed to allow customers complete customization capabilities without any requirement to update code or fork the GitHub repo. Additionally, rather than forcing customers to provide a multitude of config files for a standard or prescriptive installation, we provide and auto-deploy with Accelerator versions of most required configuration items from the reference-artifacts folder of the repo. If a customer provides the required configuration file in their Accelerator S3 input bucket, we will use the customer supplied version of the configuration file rather than the Accelerator version. At any time, either before initial installation, or in future, a customer can place new or updated SCPs, policies, or other supported file types into their input bucket and we will use those instead of or in addition to Accelerator supplied versions. Customer only need to provide the specific files they wish to override, not all files.

Customers can also define additional SCPs (or modify existing SCPs) using the name, description and filename of their choosing, and deploy them by referencing them on the appropriate organizational unit in the config file.

Prior to v1.2.5, if we updated the default files, we overwrote customers customizations during upgrade. Simply updating the timestamp *after* upgrade on the customized versions and then rerunning the state machine re-instates customer customizations. In v1.2.5 we always use the customer customized version from the S3 bucket. It's important customers assess newly provided defaults during an upgrade process to ensure they are incorporating all the latest fixes and improvements. If a customer wants to revert to Accelerator provided default files, they will need to manually copy it from the repo into their input bucket.

NOTE: Most of the provided SCPs are designed to protect the Accelerator deployed resources from modification and ensure the integrity of the Accelerator. Extreme caution must be exercised if the provided SCPs are modified. In v1.5.0 we restructured the SCPs based on a) customer requests, and b) the addition of Control Tower support for new installs.

- we reorganized and optimized our SCP's from 4 SCP files down to 3 SCP files, without removing any protections or guardrails;
- these optimizations have resulted in minor enhancements to the SCP protections and in some cases better scoping;
- the first two SCP files (Part-0 and Part-1) contain the controls which protect the integrity of the Accelerator itself;
- the third file (Sensitive, Unclass, Sandbox) contains customer data protection specific guardrails, which may change based on workload data classification or customer profiles and requirements;
- this freed the fourth SCP for use by Control Tower. As Control Tower leverages 2 SCP files on the Security OU, we have moved some of our SCP's to the account level.

2.1.9 1.1.9. I deployed AWS Managed Active Directory (MAD) as part of my deployment, how do I manage Active Directory domain users, groups, and domain policies after deployment?

Customers have clearly indicated they do NOT want to use the Accelerator to manage their Active Directory domain or change the way they manage Active Directory on an ongoing basis. Customer have also indicated, they need help getting up and running quickly. For these reasons, the Accelerator only sets the domain password policy, and creates AD users and groups on the initial installation of MAD. After the initial installation, customers must manage Windows users and groups using their traditional tools. A bastion Windows host is deployed as a mechanism to support these capabilities. Passwords for all newly created MAD users have been stored, encrypted, in AWS Secrets Manager in the Management (root) Organization AWS account.

To create new users and groups:

- RDP into the ASEA-RDGW bastion host in the Ops account
- Run ADUC and create users and groups as you please under the NETBIOSDOMAIN (example) tree
- Or run the appropriate powershell command
- Go to AWS SSO and map the Active Directory group to the appropriate AWS account and permission set

The Accelerator will not create/update/delete new AD users or groups, nor will it update the domain password policy after the initial installation of Managed Active Directory. It is your responsibility to rotate these passwords on a regular basis per your organizations password policy. (NOTE: After updating the admin password it needs to be stored back in secrets manager).

2.1.10 1.1.10. How do I suspend an AWS account?

Suspending accounts is blocked via SCP and purposely difficult, two options exist:

1. Modify SCP method (not desired)
2. Leverage the UnManaged OU
3. validate your config file contains the value: `"ignored-ous": ["UnManaged"]`
   - the state machine must be executed at least once after this value is added to the config file
4. In AWS Organizations create an OU named `UnManaged` in the root of the OU tree, if it does not exist
5. Change to the `us-east-1` region and open CloudWatch and navigate to Rules
   - Select the `PBMMAccel-MoveAccount_rule`, select actions, select `Disable`
6. In Organizations move the account to be suspended to the `UnManaged` OU
7. Change to the `us-east-1` region and open CloudWatch and navigate to Rules
   - Select the `PBMMAccel-MoveAccount_rule`, select actions, select `Enable`
8. login to the account to be suspended as the account root user
9. suspend the account through `My Account`
10. Run the state machine (from the Organization management account), the account will:
    - have a deleted=true value added to the config file
    - be moved to the suspended OU (OU value and path stays the same in the config file)
    - deleted=true causes OU validation to be skipped on this account on subsequent SM executions
11. If the AWS account was listed in the mandatory-accounts section of the config file the SM will fail (expected)
    - after the above tasks have been completed, remove all references to the suspended mandatory account from the config file
    - rerun the state machine, specifying: `{ "overrideComparison": true }`
12. Deleted accounts will continue to appear under the `Suspended` OU for 90-days

2.1.11 1.1.11. I need a new VPC, where shall I define it?

You can define a VPC in one of four major sections of the Accelerator configuration file:

- within an organization unit (this is the recommended and preferred method);
- within an account in mandatory-account-configs;
- within an account in workload-account-configs;
- defined within an organization unit, but opted-in within the account config.

We generally recommend most items be defined within organizational units, such that all workload accounts pickup their persona from the OU they are associated and minimize per account configuration. Both a local account based VPC (as deployed in the Sandbox OU accounts), or a central shared VPC (as deployed in the Dev/Test/Prod OU accounts in many of the example configs) can be defined at the OU level.

As mandatory accounts often have unique configuration requirements, for example the centralized Endpoint VPC, they must be configured within the accounts configuration. Customers can define VPC's or other account specific settings within any accounts configuration, but this requires editing the configuration file for each account configuration.

Prior to v1.5.0, local VPC's defined at the OU level were each deployed with the same CIDR ranges and therefor could not be connected to a TGW. Local VPC's requiring centralized networking (i.e. TGW connectivity) were required to be defined in each account config, adding manual effort and bloating the configuration file.

The addition of `dynamic` and `lookup` CIDR sources in v1.5.0 resolves this problem. Local VPCs can be defined in an OU, and each VPC will be dynamically assigned a unique CIDR range from the assigned CIDR pool, or looked up from the DynamoDB database. Customers can now ensure connected, templated VPCs are consistently deployed to every account in an OU, each with unique IP addresses.

v1.5.0 also added a new opt-in VPC capability. A VPC is defined in an OU and a new config file variable is added to this VPC `opt-in: true`. When opt-in is set to true, the state machine does NOT create the VPC for the accounts in the OU, essentially ignoring the VPC definition. Select accounts in the OU can then be opted-in to the VPC(s) definition, by adding the value `accountname\opt-in-vpcs: ["opt-in-vpc-name1", "opt-in-vpc-name2", "opt-in-vpc-nameN"]` to the specific accounts which need the VPC(s). A vpc definition with the specified name (i.e. `opt-in-vpc-name1`) and the value `opt-in: true`, must exist in the ou config for the specified account. When these conditions apply, the VPC will be created in the account per the OU definition. Additional opt-in VPCs can be added to an account, but VPC's cannot be removed from the opt-in-vpcs array. VPC's can be TGW attached, assuming `dynamic` cidr-src is utilized, or DynamoDB is prepopulated with the required CIDR ranges using `lookup` mode. `cidr-src` provided is suitable for disconnected Sandbox type accounts.

The Future: While Opt-In VPCs are powerful, we want to take this further. Why not deploy an AWS Service Catalog template which contains the names of all the available opt-in VPCs for the accounts OU, inside each account. An account end user could then request a new VPC for their account from the list of available opt-in patterns. A users selection would be sent to a centralized queue for approval (w/auto-approval options), which would result in the opt-in-vpc entry in that account being updated with the end users requested vpc pattern and the personalized VPC being created in the account and attached to the centralized TGW (if part of the pattern). This would ensure all VPC's conformed to a set of desirable design patterns, but also allow the end-user community choices based on their desired development and app patterns. If you like this idea, please +1 this feature request.

2.1.12 1.1.12. How do I modify and extend the Accelerator or execute my own code after the Accelerator provisions a new AWS account or the state machine executes?

Flexibility:

- The AWS Secure Environment Accelerator was developed to enable extreme flexibility without requiring a single line of code to be changed. One of our primary goals throughout the development process was to avoid making any decisions that would result in users needing to fork or branch the Accelerator codebase. This would help ensure we had a sustainable and upgradable solution for a broad customer base over time.
- Functionality provided by the Accelerator can generally be controlled by modifying the main Accelerator configuration file.
- Items like SCP's, rsyslog config, Powershell scripts, and iam-policies have config files provided and auto-deployed as part of the Accelerator to deliver on the prescriptive architecture (these are located in the \reference-artifacts folder of the Github repo for reference). If you want to alter the functionality delivered by any of these additional config files, you can simply provide your own by placing it in your specified Accelerator bucket in the appropriate sub-folder. The Accelerator will use your provided version instead of the supplied repo reference version.
- As SCP's and IAM policies are defined in the main config file, you can simply define new policies, pointing to new policy files, and provide these new files in your bucket, and they will be used.
- While a sample firewall config file is provided in the \reference-artifacts folder, it must be manually placed in your s3 bucket/folder on new Accelerator deployments
- Any/all of these files can be updated at any time and will be used on the next execution of the state machine
- Over time, we predict we will provide several sample or reference architectures and not just the current single PBMM architecture (all located in the \reference-artifacts\SAMPLE_CONFIGS folder).

Extensibility:

- Every execution of the state machine sends a state machine status event to a state machine SNS topic
- These status events include the Success/Failure status of the state machine, and on success, a list of all successfully processed AWS accounts
- While this SNS topic is automatically subscribed to a user provided email address for user notification, users can also create additional SNS subscriptions to enable triggering their own subsequent workflows, state machines, or custom code using any supported SNS subscription type (Lambda, SQS, Email, HTTPS, HTTPS)
- Additionally, objects deployed within an account have been populated in Parameter Store, see answer 1.3.2 for details

Example:

- One of our early adopter customers has developed a custom user interface which allows their clients to request new AWS environments. Clients provide items like cost center, budget, and select their environment requirements (i.e. Sandbox, Unclass or full sensitive SDLC account set). On appropriate approval, this pushes the changes to the Accelerator configuration file and triggers the state machine.
- Once the state machine completes, the SNS topic triggers their follow-up workflow, validates the requested accounts were provisioned, updates the customer's account database, and then executes a collection of customer specific follow-up workflow actions on any newly provisioned accounts.

2.1.13 1.1.13. How can I easily access my virtual machines or EC2 instances?

The preferred and recommended method to connect to instances within the Accelerator is by using AWS Systems Manager Session Manager. Session Manager allows access to instances without the need to have any open firewall ports. Session Manager allows for Command line access to instances (both Windows and Linux) directly through the AWS console, with all activity logged to CloudWatch Logs. Session Manager enables customers to connect to Windows instances with a native RDP client and Linux instances with a native SSH client, if desired. Customers can gain quick access to instances through the AWS consolve, or using their preferred clients.

General

- Both the RDGW and rsyslog instances deployed in the Ops account are properly configured for Systems Manager Session Manager
- We have implemented automation such that all instances are also automatically configured for Session Manager (i.e. configured with the appropriate launch role, has a recent session manager agent installed (most amazon ami's do), has access to an SSM endpoint)

Connecting to an Instance

- From the AWS Console
- Go to: EC2, Instances, select the instance (i.e. ASEA-RDGW), click “Connect”, select Session Manager, Connect
- Ideal for Linux or Windows Powershell users
- Everything is fully logged for future reference
- Directly through local RDP client using Session Managers tunnel capability:
- Provides direct access to your instances/host without any open internet ports on the RDGW instance, using a local/fat client tool
- Install AWS CLI v2 on your PC - available here (uninstall CLIv1 first, if installed)
- Install the SSM plugin on your PC - available here
- Get AWS temporary credentials from AWS SSO for the account your workload resides (i.e. Ops account when accessing the ASEA-RDGW instance) by selecting “Command line or programmatic access” instead of “Management Console” and paste them into a command prompt
- i.e. via logging in here: https://xxxxxxx.awsapps.com/start or
- This blog describes the process to use SSO to get credentials for the AWS CLI directly without the GUI
- Then enter: aws ssm start-session --target "i-12345678901234567" --document-name AWS-StartPortForwardingSession --parameters portNumber="3389",localPortNumber="56789"--region ca-central-1
- Command syntax is slightly different on Linux/Mac
- Replace i-1111adddce582b23c with the instance id of your RDGW instance
- A tunnel will open
- As these are tunnels to proprietary protocols (i.e. RDP/screen scraping) session content is not logged.
- Run mstsc/rdp client and connect to 127.0.0.1:56789
- By replacing 3389 with a new port for another applications (i.e. SSH running on a Linux instance), you can connect to a different application type
- You can change the local port by changing 56789 to any other valid port number (i.e. connecting to multiple instances at the same time)
- Login with the windows credentials discussed above in the format NETBIOSDOMAIN\User1 (i.e. example\user1)
- Your netbios domain is found here in your config file: "netbios-domain": "example",
- Connect to your desktop command line to command line interface of remote Windows or Linux servers, instead of through console (i.e. no tunnel):
- aws ssm start-session --target "i-090c25e64c2d9d276""--region ca-central-1
- Replace i-xxx with your instance ID
- Everything is fully logged for future reference
- If you want to remove the region from your command line, you can:
- Type: “aws configure” from command prompt, hit {enter} (key), {enter} (secret), enter: ca-central-1, {enter}

2.1.14 1.1.14. I ran the state machine but it failed when it tried to delete the default VPC? The state machine cannot delete the default VPC (Error: VPC has dependencies and cannot be deleted)

- You need to ensure that resources don’t exist in the default VPC or else the state machine won't be able to delete it. If you encounter this error, you can either delete the resources within the VPC or delete the default VPC manually and run the state machine again.

2.2 1.2. Existing Accounts/Organizations

2.2.1 1.2.1. How do I import an existing AWS account into my Accelerator managed AWS Organization (or what if I created a new AWS account with a different Organization trust role)?

- Ensure you have valid administrative privileges for the account to be invited/added

- Add the account to your AWS Organization using standard processes (i.e. Invite/Accept)
- this process does NOT create an organization trust role
- imported accounts do NOT have the quarantine SCP applied as we don't want to break existing workloads
- Login to the account using the existing administrative credentials
- Execute the Accelerator provided CloudFormation template to create the required Accelerator bootstrapping role - in the Github repo here: `reference-artifacts\Custom-Scripts\Import-Account-CFN-Role-Template.yml`
- add the account to the Accelerator config file and run the state machine
- If you simply created the account with an incorrect role name, you likely need to take extra steps:
- Update the Accelerator config file to add the parameter: `global-options\ignored-ous` = `["UnManagedAccounts"]`
- In AWS Organizations, create a new OU named `UnManagedAccounts` (case sensitive)
- Move the account to the `UnManagedAccounts` ou
- You can now remove the Quarantine SCP from the account
- Assume an administrative role into the account
- Execute the Accelerator provided CloudFormation template to create the required Accelerator bootstrapping role

2.2.2 1.2.2. Is it possible to deploy the Accelerator on top of an AWS Organization that I have already installed the AWS Landing Zone (ALZ) solution into?

Existing ALZ customers are required to uninstall their ALZ deployment before deploying the Accelerator. Please work with your AWS account team to find the best mechanism to uninstall the ALZ solution (procedures and scripts exist). Additionally, please reference section 4 of the Instation and Upgrade Guide. It may be easier to migrate AWS accounts to a new Accelerator Organization, per the process detailed in FAQ #1.2.3.

2.2.3 1.2.3. What if I want to move an account from an AWS Organization that has the ALZ deployed into an AWS Organization running the Accelerator?

Before removing the AWS account from the source organization, terminate the AWS Service Catalog product associated with the member account that you're interested in moving. Ensuring the product terminates successfully and that there aren't any remaining CloudFormation stacks in the account that were deployed by the ALZ. You can then remove the account from the existing Organization and invite it into the new organization. Accounts invited into the Organization do NOT get the `Deny All` SCP applied, as we do not want to break existing running workloads. Moving the newly invited account into its destination OU will trigger the state machine and result in the account being ingested into the Accelerator and having the guardrails applied per the target OU persona.

For a detailed procedure, please review this document.

2.3 1.3. End User Environment

2.3.1 1.3.1. Is there anything my end users need to be aware of? Why do some of my end users struggle with CloudWatch Log groups errors?

CloudWatch Log group deletion is prevented for security purposes and bypassing this rule would be a fundamental violation of security best practices. This protection does NOT exist solely to protect ASEA logs, but ALL log groups. Users of the Accelerator environment will need to ensure they set CloudFormation stack Log group retention type to RETAIN, or stack deletes will fail when attempting to delete a stack (as deleting the log group will be blocked) and users will encounter errors. As repeated stack deployments will be prevented from recreating the same log group name (as it already exists), end users will either need to check for the existence of the log group before attempting creation, or include a random hash in the log group name. The Accelerator also sets log group retention for all log groups to value(s) specified by customers in the config file and prevents end users from setting or changing Log group retentions. When creating new log groups, end users must either *not* configure a retention period, or set it to the default `NEVER expire` or they will also be blocked from creating the CloudWatch Log group. If applied by bypassing the guardrails, customer specified retention periods on log group creation will be overridden with the Accelerator specified retention period.

While a security best practice, some end users continue to request this be changed, but you need to ask: Are end users allowed to go in and clean out logs from Windows Event Viewer (locally or on domain controllers) after testing? Clean out Linux kernel logs? Apache log histories? The fundamental principal is that all and as many logs as possible will be retained for a defined retention period (some longer). In the "old days", logs

were hidden deep within OS directory structures or access restricted by IT from developers - now that we make them all centralized, visible, and accessible, end users seem to think they suddenly need to clean them up. Customers need to establish a usable and scalable log group naming standard/convention as the first step in moving past this concern, such that they can always find their active logs easily. As stated, to enable repeated install and removal of stacks during test cycles, end user CloudFormation stacks need to set log groups to RETAIN and leverage a random hash in log group naming (or check for existence, before creating).

The Accelerator provided SCPs (guardrails/protections) are our recommendations, yet designed to be fully customizable, enabling any customer to carefully override these defaults to meet their individual requirements. If insistent, we'd suggest only bypassing the policy on the Sandbox OU, and only for log groups that start with a very specific prefix (not all log groups). When a customer wants to use the delete capability, they would need to name their log group with the designated prefix - i.e. opt-in to allow CloudWatch log group deletes.

2.3.2 1.3.2. How can I leverage Accelerator deployed objects in my IaC? Do I need to manually determine the arn's and object id's of Accelerator deployed objects to leverage them in my IaC?

Objects deployed by the Accelerator which customers may need to leverage in their own IaC have been populated in parameters in AWS parameter store for use by the IaC tooling of choice. The Accelerator ensures parameters are deployed consistently across accounts and OUs, such that a customers code does not need to be updated when it is moved between accounts or promoted from Dev to Test to Prod.

Objects of the following types and their associated values are stored in parameter store: vpc, subnet, security group, elb (alb/nlb w/DNS address), IAM policy, IAM role, KMS key, ACM cert, SNS topic, and the firewall replacement variables.

Additionally, setting "populate-all-elbs-in-param-store": true for an account will populates all Accelerator wide ELB information into parameter store within that account. The sample PBMM configuration files set this value on the perimeter account, such that ELB information is available to configure centralized ingress capabilities.

2.4 1.4. Upgrades

2.4.1 1.4.1. Can I upgrade directly to the latest release, or must I perform upgrades sequentially?

Yes, currently customers can upgrade from whatever version they have deployed to the latest Accelerator version. There is no requirement to perform sequential upgrades. In fact, we strongly discourage sequential upgrades.

Given the magnitude of the v1.5.0 release, we have added a one-time requirement that all customers upgrade to a minimum of v1.3.8 before attempting to upgrade to v1.5.0.

2.4.2 1.4.2. Why do I get the error "There were errors while comparing the configuration changes:" when I update the config file?

In v1.3.0 we added protections to allow customers to verify the scope of impact of their intended changes to the configuration file. In v1.3.0 and above, the state machine does not allow changes to the config file (other than new accounts) without providing the `scope` parameter. Please refer to section 1.1 of the `State Machine behavior and inputs` Guide for more details.

2.5 1.5. Support Concerns

2.5.1 1.5.1. The Accelerator is written in CDK and deploys CloudFormation, does this restrict the Infrastructure as Code (IaC) tools that I can use?

No. Customers can choose the IaC framework or tooling of their choice. The tooling used to deploy the Accelerator has no impact on the automation framework customers use to deploy their applications within the Accelerator environment. It should be noted that the functionality deployed by the Accelerator is extremely platform specific and would not benefit from multi-platform IaC frameworks or tooling.

2.5.2 1.5.2. What happens if AWS stops enhancing the Accelerator?

The Accelerator is an open source project, should AWS stop enhancing the solution for any reason, the community has access to the full codebase, its roadmap and history. The community can enhance, update, fork and take ownership of the project, as appropriate.

The Accelerator is an AWS CDK based project and synthesizes to native AWS CloudFormation. AWS sub-accounts simply contain native CloudFormation stacks and associated custom resources, when required. The Accelerator architecture is such that all CloudFormation stacks are native to each AWS account with no links or ties to code in other AWS accounts or even other stacks within the same AWS account. This was an important initial design decision.

The Accelerator codebase can be completely uninstalled from the organization management (root) account, without any impact to the deployed functionality or guardrails. In this situation, guardrail updates and new account provisioning reverts to a manual process. Should a customer decide they no longer wish to utilize the solution, they can remove the Accelerator codebase without any impact to deployed resources and go back to doing things natively in AWS as they did before they deployed the Accelerator. By adopting the Accelerator, customers are not locking themselves in or making a one-way door decision.

2.5.3 1.5.3. What level of Support will the ASEA have from AWS Support?

The majority of the solution leverages native AWS services which are fully supported by AWS Support. Additionally, the Accelerator is an AWS CDK based project and synthesizes to native AWS CloudFormation. AWS sub-accounts simply contain native CloudFormation stacks and associated custom resources (when required). The Accelerator architecture is such that all CloudFormation stacks are native to each AWS account with no direct links or ties to code in other AWS accounts (no stacksets, no local CDK). This was an important project design decision, keeping deployed functionality in independent local CloudFormation stacks and decoupled from solution code, which allows AWS support to effectively troubleshoot and diagnose issues local to the sub-account.

As the Accelerator also includes code, anything specifically related to the Accelerator codebase will be only supported on a "best effort" basis by AWS support, as AWS support does not support custom code. The first line of support for the codebase is typically your local AWS team (your SA, TAM, Proserve and/or AWS Partner). As an open source project, customers can file requests using GitHub Issues against the Accelerator repository or open a discussion in GitHub discussions. Most customer issues arise during installation and are related to configuration customization or during the upgrade process.

2.5.4 1.5.4. What does it take to support the Accelerator?

We advise customers to allocate a 1/2 day per quarter to upgrade to the latest Accelerator release.

Customers have indicated that deploying the Accelerator reduces their ongoing operational burden over operating in native AWS, saving hours of effort every time a new account is provisioned by automating the deployment of the persona associated with new accounts (guardrails, networking and security). The Accelerator does NOT alleviate a customers requirement to learn to effectively operate in the cloud (like monitoring security tooling/carrying out Security Operation Center (SOC) duties). This effort exists regardless of the existence of the Accelerator.

2.5.5 1.5.5. Is the Accelerator only designed and suitable for Government of Canada or PBMM customers?

No. The Accelerator is targeted at ***any AWS customer*** that is looking to automate the deployment and management of a comprehensive end-to-end multi-account environment in AWS. It is ideally suited for customers interested in achieving a high security posture in AWS.

The Accelerator is a sophisticated deployment framework that allows for the deployment and management of virtually any AWS multi-account "Landing Zone" architecture without any code modifications. The Accelerator is actually delivering two separate and distinct products which can each be used on their own:

1. the Accelerator the tool, which can deploy virtually any architecture based on a provided config file (no code changes), and;
2. the Government of Canada (GC) prescriptive PBMM architecture which is delivered as a sample configuration file and documentation.

The tooling was purposely built to be extremely flexible, as we realized that some customers may not like some of the opinionated and prescriptive design decisions we made in the GC architecture. Virtually every feature being deployed can be turned on/off, not be used or can have its configuration adjusted to meet your specific design requirements.

We are working on building a library of sample config files to support additional customer needs and better demonstrate product capabilities and different architecture patterns. In no way is it required that the prescriptive GC architecture be used or deployed. Just because we can deploy, for example, an AWS Managed Active Directory, does not mean you need to use that feature of the solution. Disabling or changing these capabilities also requires zero code changes.

While the prescriptive sample configuration files were originally developed based on GC requirements, they were also developed following AWS Best Practices. Additionally, many security frameworks around the world have similar and overlapping security requirements (you can only do security so many ways). The provided architecture is applicable to many security compliance regimes around the world and not just the GC.

2.6 1.6. Deployed Functionality

2.6.1 1.6.1. I wish to be in compliance with the 12 GC TBS Guardrails, what don't you cover with the provided sample architecture?

The AWS SEA allows for a lot of flexibility in deployed architectures. If used, the provided PBMM sample architecture was designed to help deliver on the technical portion of *all* 12 of the GC guardrails, when automation was possible.

What don't we cover? Assigning MFA to users is a manual process. Specifically you need to procure Yubikeys for your root/break glass users, and enable a suitable form of MFA for *all* other users (i.e. virtual, email, other). The guardrails also include some organizational processes (i.e. break glass procedures, or signing an MOU with CCCS) which customers will need to work through independently.

While AWS is providing the tools to help customer be compliant with the 12 PBMM guardrails (which were developed in collaboration with the GC) - it's up to each customers ITSec organization to assess and determine if the deployed controls actually meet their security requirements.

Finally, while we started with a goal of delivering on the 12 guardrails, we believe we have extended well beyond those security controls, to further help customers move towards meeting the full PBMM technical control profile (official documentation is weak in this area at this time).

2.6.2 1.6.2. Does the ALB perform SSL offloading?

As configured - the perimeter ALB decrypts incoming traffic using its certificate and then re-encrypts it with the certificate for the back-end ALB. The front-end and back-end ALB's can use the same or different certs. If the Firewall needs to inspect the traffic, it also needs the backend certificate be manually installed.

2.6.3 1.6.3. What is the recommended approach to manage the ALB certificates deployed by the Accelerator?

The Accelerator installation process allows customers to provide their own certificates (either self-signed or generated by a CA), to enable quick and easy installation and allowing customers to test end-to-end traffic flows. After the initial installation, we recommend customers leverage AWS Certificate Manager (ACM) to easily provision, manage, and deploy public and private SSL/TLS certificates. ACM helps manage the challenges of maintaining certificates, including certificate rotation and renewal, so you don’t have to worry about expiring certificates.

The Accelerator provides 3 mechanisms to enable utilizing certificates with ALB's:

- **Method 1** - IMPORT a certificate into AWS Certificate Manager from a 3rd party product
- When using a certificate that does not have a certificate chain (usually this is the case with Self-Signed)

```
json
    "certificates": [
        {
          "name": "My-Cert",
          "type": "import",
          "priv-key": "certs/example1-cert.key",
          "cert": "certs/example1-cert.crt"
        }
      ]
```

- When using a certificate that has a certificate chain (usually this is the case when signed by a Certificate Authority with a CA Bundle)

```
json
    "certificates": [
        {
          "name": "My-Cert",
```

```
            "type": "import",
            "priv-key": "certs/example1-cert.key",
            "cert": "certs/example1-cert.crt",
            "chain": "certs/example1-cert.chain"
        }
    ]
```

- this mechanism allows a customer to generate certificates using their existing tools and processes and import 3rd party certificates into AWS Certificate Manager for use in AWS
- Self-Signed certificates should NOT be used for production (samples were provided simply to demonstrate functionality)
- both a `.key` and a `.crt` file must be supplied in the customers S3 input bucket
- "cert" must contain only the certificate and not the full chain
- "chain" is an optional attribute that contains the certificate chain. This is generally used when importing a CA signed certificate
- this will create a certificate in ACM and a secret in secrets manager named `accelerator/certificates/My-Cert` in the specified AWS account(s), which points to the newly imported certificates ARN
- **Method 2** - REQUEST AWS Certificate Manager generate a certificate

```
json
  "certificates": [
    {
      "name": "My-Cert",
      "type": "request",
      "domain": "*.example.com",
      "validation": "DNS",
```

```
      "san": ["www.example.com"]
    }
  ]
```

- this mechanism allows a customer to generate new public certificates directly in ACM
- both `DNS` and `EMAIL` validation mechanisms are supported (DNS recommended)
- this requires a ***Public*** DNS zone be properly configured to validate you are legally entitled to issue certificates for the domain
- this will also create a certificate in ACM and a secret in secrets manager named `accelerator/certificates/My-Cert` in the specified AWS account(s), which points to the newly imported certificates ARN
- this mechanism should NOT be used on new installs, skip certificate and ALB deployment during initial deployment (removing them from the config file) and simply add on a subsequent state machine execution
- Process:
- you need a public DNS domain properly registered and configured to publicly resolve the domain(s) you will be generating certificates for (i.e. example.com)
- domains can be purchased and configured in Amazon Route53 or through any 3rd party registrar and DNS service provider
- in Accelerator phase 1, the cert is generated, but the stack does NOT complete deploying (i.e. it waits) until certificate validation is complete
- during deployment, go to the AWS account in question, open ACM and the newly requested certificate. Document the authorization CNAME record required to validate certificate generation
- add the CNAME record to the zone in bullet 1 (in Route53 or 3rd party DNS provider) (documented here)
- after a few minutes the certificate will validate and switch to `Issued` status
- Accelerator phase 1 will finish (as long as the certificate is validated before the Phase 1 credentials time-out after 60-minutes)
- the ALB will deploy in a later phase with the specified certificate
- **Method 3** - Manually generate a certificate in ACM
- this mechanism allows a customer to manually generate certificates directly in the ACM interface for use by the Accelerator
- this mechanism should NOT be used on new installs, skip certificate and ALB deployment during initial deployment (removing them from the config file) and simply add on a subsequent state machine execution
- Process:
- go to the AWS account for which you plan to deploy an ALB and open ACM
- generate a certificate, documenting the certificates ARN
- open Secrets manager and generate a new secret of the format `accelerator/certificates/My-Cert` (of type `Plaintext` under `Other type of secrets`), where `My-Cert` is the unique name you will use to reference this certificate
- In all three mechanisms a secret will exist in Secrets Manager named `accelerator/certificates/My-Cert` which contains the ARN of the certificate to be used.
- In the Accelerator config file, find the definition of the ALB for that AWS account and specify `My-Cert` for the ALB `cert-name`

```
"alb": [
  {
    "cert-name": "My-Cert"
  }
]
```

- The state machine will fail if you specify a certificate in any ALB which is not defined in Secrets Manager in the local account.

We suggest the most effective mechanism for leveraging ACM is by adding CNAME authorization records to the relevant DNS domains using Method 2, but may not appropriate right for all customers.

2.6.4 1.6.4. Why do we have rsyslog servers? I thought everything was sent to CloudWatch?

The rsyslog servers are included to accept logs for appliances and third party applications that do not natively support the CloudWatch Agent from any account within a customers Organization. These logs are then immediately forwarded to CloudWatch Logs within the account the rsyslog servers are deployed (Operations) and are also copied to the S3 immutable bucket in the log-archive account. Logs are only persisted on the rsyslog hosts for 24 hours. The rsyslog servers are required to centralize the 3rd party firewall logs (Fortinet Fortigate).

2.6.5 1.6.5. Can you deploy the solution without Fortinet Firewall Licenses?

Yes, if license files are not provided, the firewalls will come up configured and route traffic, but customers will have no mechanism to manage the firewalls/change the configuration until a valid license file is added. If invalid licence files are provided, the firewalls will fail to load the provided configuration, will not enable routing, will not bring up the VPN tunnels and will not be manageable. Customers will need to either remove and redeploy the firewalls, or manually configure them. If performing a test deployment, please work with your local Fortinet account team to discuss any options for temporary evaluation licenses.

Additionally, several additional firewall options are now available, including using AWS Network Firewall, a native AWS service.

2.6.6 1.6.6. I installed additional software on my Accelerator deployed RDGW / rsyslog host, where did it go?

The RDGW and rsyslog hosts are members of auto-scaling groups. These auto-scaling groups have been configured to refresh instances in the pool on a regular basis (7-days in the current sample config files). This ensures these instances are always clean. Additionally, on every execution of the Accelerator state machine the ASG are updated to the latest AWS AMI for the instances. When the auto-scaling group refreshes its instances, they will be redeployed with the latest patch release of the AMI/OS. It is recommended that the state machine be executed monthly to ensure the latest AMI's are always in use.

Customers wanting to install additional software on these instances should either a) update the automated deployment scripts to install the new software on new instance launch, or b) create and specify a custom AMI in the Accelerator configuration file which has the software pre-installed ensuring they are also managing patch compliance on the instance through some other mechanism.

At any time, customers can terminate the RDGW or rsyslog hosts and they will automatically be re-created from the base images with the latest patch available at the time of the last Accelerator State Machine execution.

2.6.7 1.6.7. Some sample configurations provide NACLs and Security Groups. Is that enough?

Security group egress rules are often used in 'allow all' mode (`0.0.0.0/0`), with the focus primarily being on consistently allow listing required ingress traffic (centralized ingress/egress controls are in-place using the perimeter firewalls). This ensures day to day activities like patching, access to DNS, or to directory services access can function on instances without friction.

The Accelerator provided sample security groups in the workload accounts offer a good balance that considers both security, ease of operations, and frictionless development. They allow developers to focus on developing, enabling them to simply use the pre-created security constructs for their workloads, and avoid the creation of wide-open security groups. Developers can equally choose to create more appropriate least-privilege security groups more suitable for their application, if they are skilled in this area. It is expected as an application is promoted through the SDLC cycle from Dev through Test to Prod, these security groups will be further refined by the extended customers teams to further reduce privilege, as appropriate. It is expected that each customer will review and tailor their Security Groups based on their own security requirements. The provided security groups ensures day to day activities like patching, access to DNS, or to directory services access can function on instances without friction, with the understanding further protections are providing by the central ingress/egress firewalls.

The use of NACLs are general discouraged, but leveraged in this architecture as a defense-in-depth mechanism. Security groups should be used as the primary access control mechanism. As with security groups, we encourage customers to review and tailor their NACLs based on their own security requirements.

2.6.8 1.6.8. Can I deploy the solution as the account root user?

No, you cannot install as the root user. The root user has no ability to assume roles which is a requirement to configure the sub-accounts and will prevent the deployment. As per the installation instructions, you require an IAM user with the `AdministratorAccess` policy attached.

2.6.9 1.6.9. Is the Organizational Management root account monitored similarly to the other accounts in the organization?

Yes, all accounts including the Organization Management or root account have the same monitoring and logging services enabled. When supported, AWS security services like GuardDuty, Macie, and Security Hub have their delegated administrator account configured as the "security" account. These tools can be used within each local account (including the Organization Management account) within the organization to gain account level visibility or within the Security account for Organization wide visibility. For more information about monitoring and logging refer to architecture documentation.

2.6.10 1.6.10. How are the perimeter firewall configurations and licensing managed after deployment?

While you deploy the perimeter firewalls with the Accelerator you will continue to manage firewall updates, configuration changes, and license renewals from the respective firewall management interface and not from the Accelerator config file. As these changes are not managed by the Accelerator you do not need to rerun the state machine to implement or track any of these changes. You can update the AMI of the 3rd party firewalls using the Accelerator, you must first remove the existing firewalls and redeploy them (as the Elastic IP's (EIP's) will block a parallel deployment) or deploy a second parallel firewall cluster and de-provision the first cluster when ready.

2.6.11 1.6.11. Can the Fortinet Firewall deployments use static private IP address assignments?

Yes, the `"port"` stanza in the configuration file can support a private static IP address assignment from the az and subnet. Care must be exercised to assure the assigned IP address is within the correct subnet and availability zone. Consideration must also be given to the Amazon reserved IP addresses (first three addresses, and the last) within subnets when choosing an IP Address to assign.

Using the `config.example.json` as a reference, static IP Assignments would look like this in the `ports:` stanza of the firewall deployment.

```
"ports": [
  {
    "name": "Public",
    "subnet": "Public",
    "create-eip": true,
    "create-cgw": true,
    "private-ips": [
      {
        "az": "a",
        "ip": "100.96.250.4"
      },
      {
        "az": "b",
        "ip": "100.96.250.132"
      }
    ]
  },
  {
    "name": "OnPremise",
    "subnet": "OnPremise",
    "create-eip": false,
    "create-cgw": false,
    "private-ips": [
      {
        "az": "a",
        "ip": "100.96.250.68"
      },
      {
        "az": "b",
        "ip": "100.96.250.196"
      }
    ]
  }
  ...
],
```

Where `private-ips` are not present for the subnet or availability zone an address will be assigned automatically from available addresses when the firewall instance is created.

2.6.12 1.6.12. I've noticed CloudTrail logs and in certain situation VPC flow logs are stored in the centralized log-archive account logging bucket twice?

Yes. Cloudtrail is configured to send its logs directly to S3 for centralized immutable log retention. CloudTrail is also configured to send it's logs to a centralized Organizational CloudWatch Log group such that the trail can be a) easily queried online using CloudWatch Insights across all AWS accounts in the organization, and b) to enable alerting based on undesirable API activity using CloudWatch Metrics and Alarms. All CloudWatch Log groups are also configured to be sent, using Amazon Kinesis, to S3 for centralized immutable log retention.

VPC flow log destinations can be configured in the config file. The example config files are set to send the VPC flow logs to both S3 and CloudWatch Logs by default for the same reasons as CloudTrail.

To reduce the duplicate long-term storage of these two specific CloudWatch Log types, customers can set `cwl-glbl-exclusions` under `central-log-services` to: `["/${ACCELERATOR_PREFIX_ND}/flowlogs/*", "/${ACCELERATOR_PREFIX_ND}/CloudTrail*"]` to prevent these specifically named log groups from being stored on S3. This setting also prevents the Accelerator from setting the customer desired log group retention period defined in the config file, once implemented, for those log groups. Therfore, we do not recommend this exception be applied during the initial installation, as the retention setting on these CWL groups will remain the default (infinite). If `cwl-glbl-exclusions` is set after initial install, the defined retention will be configured during install and will remain set to the value present when the exception was applied to those log groups. This allows logs to be stored in CloudWatch Logs for quick and easy online access (short-retention only), and stored in S3 for long-term retention and access.

Side note: CloudTrail S3 data plane logs are enabled at the Organizational level, meaning all S3 bucket access is logged. As CloudTrail is writing to a bucket within the Organization, Cloudtrail itself is accessing the bucket, seemingly creating a cyclical loop. As CloudTrail writes to S3 in 5-10min batches, Cloudtrail will actually only cause one extra log 'entry' every 5-10minutes and not per s3 event, mitigating major concerns. Today, with an Organization trail logging data plane events for all buckets - there is no way to exclude any one bucket. But - having clear view of who accessed/changed logs, including AWS services, is important.

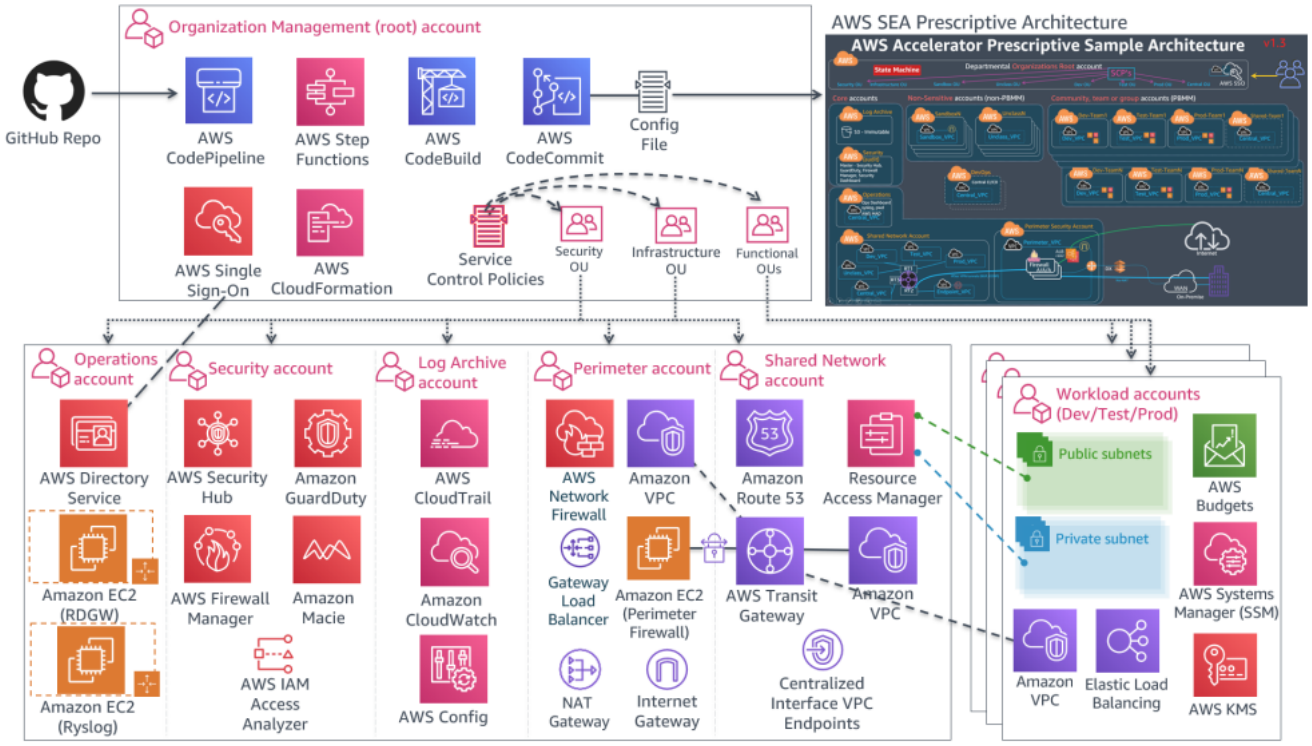
...Return to Accelerator Table of Contents

3. AWS Secure Environment Accelerator

The AWS Accelerator is a tool designed to help deploy and operate secure multi-account, multi-region AWS environments on an ongoing basis. The power of the solution is the configuration file that drives the architecture deployed by the tool. This enables extensive flexibility and for the completely automated deployment of a customized architecture within AWS without changing a single line of code.

While flexible, the AWS Accelerator is delivered with a sample configuration file which deploys an opinionated and prescriptive architecture designed to help meet the security and operational requirements of many governments around the world. Tuning the parameters within the configuration file allows for the deployment of customized architectures and enables the solution to help meet the multitude of requirements of a broad range of governments and public sector organizations.

The installation of the provided prescriptive architecture is reasonably simple, deploying a customized architecture does require extensive understanding of the AWS platform. The sample deployment specifically helps customers meet NIST 800-53 and/or CCCS Medium Cloud Control Profile (formerly PBMM).



3.1 What specifically does the Accelerator deploy and manage?

A common misconception is that the AWS Secure Environment Accelerator only deploys security services, not true. The Accelerator is capable of deploying a complete end-to-end hybrid enterprise multi-region cloud environment.

Additionally, while the Accelerator is initially responsible for deploying a prescribed architecture, it more importantly allows for organizations to operate, evolve, and maintain their cloud architecture and security controls over time and as they grow, with minimal effort, often using native AWS tools. Customers don't have to change the way they operate in AWS.

Specifically the accelerator deploys and manages the following functionality, both at initial accelerator deployment and as new accounts are created, added, or onboarded in a completely automated but customizable manner:

3.1.1 Creates AWS Account

- Core Accounts - as many or as few as your organization requires, using the naming you desire. These accounts are used to centralize core capabilities across the organization and provide `Control Panel` like capabilities across the environment. Common core accounts include:
- Shared Network
- Operations
- Perimeter
- Log Archive
- Security Tooling
- Workload Accounts - automated concurrent mass account creation or use AWS organizations to scale one account at a time. These accounts are used to host a customer's workloads and applications.
- Scalable to 1000's of AWS accounts
- Supports AWS Organizations nested ou's and importing existing AWS accounts
- Performs 'account warming' to establish initial limits, when required
- Automatically submits limit increases, when required (complies with initial limits until increased)
- Leverages AWS Control Tower **(NEW)**

3.1.2 Creates Networking

- Transit Gateways and TGW route tables (incl. inter-region TGW peering)
- Centralized and/or Local (bespoke) VPC's
- Subnets, Route tables, NACLs, Security groups, NATGWs, IGWs, VGWs, CGWs
- VPC Endpoints (Gateway and Interface, Centralized or Local)
- Route 53 Private and Public Zones, Resolver Rules and Endpoints, VPC Endpoint Overloaded Zones
- All completely and individually customizable (per account, VPC, subnet, or OU)
- Layout and customize your VPCs, subnets, CIDRs and connectivity the way you want
- Static or Dynamic **(NEW)** VPC and subnet CIDR assignments
- Deletes default VPC's (worldwide)
- AWS Network Firewall **(NEW)**

3.1.3 Cross-Account Object Sharing

- VPC and Subnet sharing, including account level re-tagging (Per account security group 'replication')
- VPC attachments and peering (local and cross-account)
- Zone sharing and VPC associations
- Managed Active Directory sharing, including R53 DNS resolver rule creation/sharing
- Automated TGW inter-region peering
- Populate Parameter Store with all `user` objects to be used by customers' IaC
- Deploy and share SSM documents (4 provided out-of-box, ELB Logging, S3 Encryption, Instance Profile remediation, Role remediation)
- customer can provide their own SSM documents for automated deployment and sharing

3.1.4 Identity

- Creates Directory services (Managed Active Directory and Active Directory Connectors)
- Creates Windows admin bastion host auto-scaling group
- Set Windows domain password policies

- Set IAM account password policies
- Creates Windows domain users and groups (initial installation only)
- Creates IAM Policies, Roles, Users, and Groups
- Fully integrates with and leverages AWS SSO for centralized and federated login

3.1.5 Cloud Security Services

- Enables and configures the following AWS services, worldwide w/central designated admin account:
- Guardduty w/S3 protection
- Security Hub (Enables designated security standards, and disables individual controls)
- Firewall Manager
- CloudTrail w/Insights and S3 data plane logging
- Config Recorders/Aggregator
- Conformance Packs and Config rules (95 out-of-box NIST 800-53 rules, 2 custom rules, customizable per OU)
- Macie
- IAM Access Analyzer
- CloudWatch access from central designated admin account (and setting Log group retentions)

3.1.6 Other Security Capabilities

- Creates, deploys and applies Service Control Policies
- Creates Customer Managed KMS Keys (SSM, EBS, S3), EC2 key pairs, and secrets
- Enables account level default EBS encryption and S3 Block Public Access
- Configures Systems Manager Session Manager w/KMS encryption and centralized logging
- Creates and configures AWS budgets (customizable per ou and per account)
- Imports or requests certificates into AWS Certificate Manager
- Deploys both perimeter and account level ALB's w/Lambda health checks, certificates and TLS policies
- Deploys & configures 3rd party firewall clusters and management instances (leverages marketplace)
- Gateway Load Balancer w/auto-scaling **(NEW)** and VPN IPSec BGP ECMP deployment options
- Protects Accelerator deployed and managed objects
- Sets Up SNS Alerting topics (High, Medium, Low, Blackhole priorities)
- Deploys CloudWatch Log Metrics and Alarms
- Deploys customer provided custom config rules (1 provided out-of-box, No EC2 Instance Profile)

3.1.7 Centralized Logging and Alerting

- Deploys an rsyslog auto-scaling cluster behind a NLB, all syslogs forwarded to CloudWatch Logs
- Centralized access to "Cloud Security Service" Consoles from designated AWS account
- Centralizes logging to a single centralized S3 bucket (enables, configures and centralizes)
- VPC Flow logs w/Enhanced metadata fields (also sent to CWL)
- Organizational Cost and Usage Reports
- CloudTrail Logs including S3 Data Plane Logs (also sent to CWL)
- All CloudWatch Logs (includes rsyslog logs)
- Config History and Snapshots
- Route 53 Public Zone Logs (also sent to CWL)

- GuardDuty Findings
- Macie Discovery results
- ALB Logs
- SSM Session Logs (also sent to CWL)
- Resolver Query Logs (also sent to CWL)
- Email alerting for CloudTrail Metric Alarms, Firewall Manager Events **(NEW)**, Security Hub Findings incl. Guardduty Findings **(NEW)**

3.2 Relationship with AWS Landing Zone Solution (ALZ)

The ALZ was an AWS Solution designed to deploy a multi-account AWS architecture for customers based on best practices and lessons learned from some of AWS' largest customers. The AWS Accelerator draws on design patterns from the Landing Zone, and re-uses several concepts and nomenclature, but it is not directly derived from it, nor does it leverage any code from the ALZ. The Accelerator is a standalone solution with no dependence on ALZ.

3.3 Relationship with AWS Control Tower

The AWS Secure Environment Accelerator now leverages AWS Control Tower! **(NEW)**

With the release of v1.5.0, the AWS Accelerator adds the capability to be deployed on top of AWS Control Tower. Customers get the benefits of the fully managed capabilities of AWS Control Tower combined with the power and flexibility of the Accelerators Networking and Security orchestration.

3.4 Accelerator Installation Process (Summary)

This summarizes the installation process, the full installation document can be found in the documentation section below.

- Create a config.json (or config.yaml) file to represent your organizations requirements (several samples provided)
- Create a Secrets Manager Secret which contains a GitHub token that provides access to the Accelerator code repository
- Create a unique S3 input bucket in the management account of the region you wish to deploy the solution and place your config.json and any additional custom config files in the bucket
- Download and execute the latest release installer CloudFormation template in your management accounts preferred 'primary' / 'home' region
- Wait for:
- CloudFormation to deploy and start the Code Pipeline (~5 mins)
- Code Pipeline to download the Accelerator codebase and install the Accelerator State Machine (~10 mins)
- The Accelerator State Machine to finish execution (~1.25 hrs Standalone version, ~2.25 hrs Control Tower Version)
- Perform required one-time post installation activities (configure AWS SSO, set firewall passwords, etc.)
- On an ongoing basis:
- Use AWS Organizations to create new AWS accounts, which will automatically be guardrailed by the Accelerator
- Update the config file in CodeCommit and run the Accelerator State Machine to:
- deploy, configure and guardrail multiple accounts at the same time (~25 min Standalone, ~50 min/account Control Tower)
- change Accelerator configuration settings (~25 min)

4. Installation & Upgrades

4.1 1. Accelerator Installation and Upgrade Guide

We encourage customers installing the Accelerator to get the support of their local AWS account team (SA, TAM, CSM, Proserve) to assist with the installation of the Accelerator, as the Accelerator leverages, deploys, or orchestrates over 30 different AWS services.

Users are strongly encouraged to also read the Accelerator Operations/Troubleshooting Guide before installation. The Operations/Troubleshooting Guide provides details as to what is being performed at each stage of the installation process, including detailed troubleshooting guidance.

These installation instructions assume one of the prescribed architectures is being deployed.

4.1.1 2. Installation

4.1.2 2.1. Prerequisites

2.1.1. General

- Management or root AWS Organization account (the AWS Accelerator cannot be deployed in an AWS sub-account)
- No additional AWS accounts need to be pre-created before Accelerator installation
- If required, a limit increase to support your desired number of new AWS sub-accounts (default limit is 10 sub-accounts)
- Valid Accelerator configuration file, updated to reflect your requirements (see below)
- Determine your primary or Accelerator `control` or `home` region, this is the AWS region in which you will most often operate
- Government of Canada customers are still required to do a standalone installation at this time, please request standalone installation instructions from your Account SA or TAM
- The Accelerator *can* be installed into existing AWS Organizations - see caveats and notes in section 4 below
- Existing AWS Landing Zone Solution (ALZ) customers are required to remove their ALZ deployment before deploying the Accelerator. Scripts are available to assist with this process.
- Changes to the Accelerator codebase are strongly discouraged unless they are contributed and accepted back to the solution. Code customization will block the ability to upgrade to the latest release and upgrades are encouraged to be done between quarterly to semi-annually. The solution was designed to be extremely customizable without changing code, existing customers following these guidelines have been able to upgrade across more than 50 Accelerator releases, while maintaining their customizations and gaining the latest bug fixes, features and enhancements without any developer or professional services based support. Please see this FAQ for more details.

4.1.3 2.2. Production Deployment Planning

2.2.1. General

For any deployment of the Accelerator which is intended to be used for production workloads, you must evaluate all these decisions carefully. Failure to understand these choices could cause challenges down the road. If this is a "test" or "internal" deployment of the Accelerator which will not be used for production workloads, you can leave the default config values.

2.2.2. OU Structure Planning

Plan your OU and core account structure carefully. By default, we suggest: `Security, Infrastructure, Central, Sandbox, Dev, Test, Prod`.

- The `Security` OU will contain the `Security` account, the `Log Archive` account, and the Organization `Management` account.
- The `Infrastructure` OU will hold the remainder of the accounts shared or utilized by the rest of the organization (`Shared Network`, `Perimeter`, and `Operations`).
- The remainder of the OUs correspond with major permission shifts in the SDLC cycle and NOT every stage an organization has in their SDLC cycle (i.e. QA or pre-prod would be included in one of the other OUs).
- The `Central` OU is used to hold accounts with workloads shared across Dev, Test, and Prod environments like centralized CI/CD tooling.
- The v1.5.0 release aligns the Accelerator OU and account structure with AWS multi-account guidance, splitting the `core` OU into the `Security` and `Infrastructure` OUs.

Note: While OUs can be renamed or additional OUs added at a later point in time, deployed AWS accounts CANNOT be moved between top-level OUs (guardrail violation), nor can top-level OUs easily be deleted (requires deleting all AWS accounts from within the OU first).

2.2.3. Network Configuration Planning

If deploying the prescriptive architecture using the Full or Lite sample config files, you will need the following network constructs:

1. Six (6) RFC1918 Class B address blocks (CIDR's) which do not conflict with your on-premise networks (a single /13 block works well)
2. VPC CIDR blocks cannot be changed after installation, this is simply the way the AWS platform works, given everything is built on top of them. Carefully consider your address block selection.
3. one block for each OU, except Sandbox which is not routable (Sandbox OU will use a 7th non-routed address block)
4. the "core" Class B range will be split to support the Endpoint VPC and Perimeter VPC (with extra addresses remaining for future use)
5. Given a shared VPC architecture is leveraged (prevents stranded islands of CIDR blocks and reduces networking costs), we have assigned a class B address block to each VPC to future proof the deployment. Smaller customers can successfully deploy with a half class B CIDR block per shared VPC.
6. Two (2) RFC6598 /23 address blocks (Government of Canada (GC) requirement only)
7. Used for AWS Managed Active Directory (MAD) deployment and perimeter underlay network
8. non-GC customers can replace the RFC6598 address space with the extra unused addresses from the above RFC1918 CIDR range above (the App2 subnets in the Central VPC and the Perimeter VPC address space)
9. BGP ASN's for network routing, one for each of:
10. Transit Gateway (one unique ASN per TGW, multi-region example requires a second ASN)
11. IPSec VPN Firewall Cluster (if deployed)
12. VGW for DirectConnect connectivity (only shown in the config.multi-region-example.json)
13. For example: the Control Tower with Network Firewall example config requires a single BGP ASN for the TGW, the IPSec VPN example requires two BGP ASN's, and the multi-region example requires five unique BGP ASN's.

NOTE: Prior to v1.5.0 CIDR ranges were assigned to each VPC and subnet throughout the config file. This required customers to perform extensive updates across the config file when needing to move to specific IP ranges compatible with a customer's existing on-premise networks. While this is still supported for those wanting to control exactly what address is used on every subnet, the solution has added support for dynamic CIDR assignments and the sample config files have been updated to reflect. New installs will have CIDR's pulled from CIDR pools, defined in the global-options section of the config file with state maintained in DynamoDB. The v1.5.0 custom upgrade guide will provides details on the upgrade process and requirements to migrate to the new CIDR assignment system, if desired. A script was created to assist with this migration.

2.2.4. DNS, Domain Name, TLS Certificate Planning

If deploying the prescriptive architecture, you must decide on:

1. A unique Windows domain name (`organizationaws` / `organization.aws` , `organizationcloud` / `organization.cloud` , etc.). Given this is designed as the primary identity store and used to domain join all cloud hosted workloads, changing this in future is difficult. Pick a Windows domain name that does NOT conflict with your on-premise AD domains, ensuring the naming convention conforms to your organizations domain naming standards to ensure you can eventually create a domain trust between the MAD and on-premise domains/forests
2. DNS Domain names and DNS server IP's for on-premise private DNS zones requiring cloud resolution (can be added in future)
3. DNS Domain for a cloud hosted public zone `"public": ["organization.cloud-nuage.canada.ca"]` (can be added in future)
4. DNS Domain for a cloud hosted private zone `"private": ["organization.cloud-nuage.gc.ca"]` (can be added in future)
5. Wildcard TLS certificate for each of the 2 previous zones (can be added/changed in future)

2.2.5. Email Address Planning

1. While you require a minimum of 6 ***unique*** email addresses (1 per sub-account being created), we recommend at least 20 ***unique*** email ALIASES associated with a single mailbox, never used before to open AWS accounts, such that you do not need to request new email aliases every time you need to create a new AWS account and they can all be monitored via a single mailbox. These email addresses can ***never*** have been used to previously open an AWS account.
2. You additionally require email addresses for the following additional purposes (these can be existing monitored mailboxes and do not need to be unique):
3. Accelerator execution (state machine) notification events (1 address)
4. High, Medium and Low security alerts (3 addresses if you wish to segregate alerts)
5. Budget notifications

2.2.6. Centralized Ingress/Egress Firewalls

As of v1.5.0 the Accelerator offers multiple automated firewall deployment options:

a) AWS Network Firewall (native AWS Cloud service)

- Defined in the config file as part of a VPC

b) 3rd party firewalls interconnected to the cloud tenancy via IPSec VPN (Active/Active using BGP + ECMP)

- Defined in the config file under deployments w/TGW VPN attachments
- this was the only automated option prior to v1.5.0
- a sample Fortinet Fortigate configuration is provided (both PAYGO and BYOL supported)
- For Fortinet BYOL, requires minimum 2 valid license files (evaluation licenses adequate) (can be added in future)

c) 3rd party firewalls interconnected to the cloud tenancy via Gateway Load Balancer (GWLB) in an auto-scaling group

- Defined in the config file under both deployments and load balancers
- a sample Checkpoint CloudGuard configuration is provided (both PAYGO and BYOL supported)

d) Customer gateway (CGW) creation, to enable connectivity to on-premises firewalls or manually deployed cloud firewalls

- Defined in the config file under deployments w/TGW VPN attachments (but without an AMI or VPC association)

Examples of each of the firewall options have been included as variants of the Lite config file example.

Note: While we only provide a single example for each 3rd party implementation today, the implementations are generic and should be usable by any 3rd party firewall vendor, assuming they support the required features and protocols. The two examples were driven by customer demand and heavy lifting by the 3rd party vendor. We look forward to additional vendors developing and contributing additional sample configurations. For new 3rd party integrations, we encourage the use of the GWLB approach.

2.2.7. Other

1. We recommend installing with the default Accelerator Name (`ASEA`) and Accelerator Prefix (`ASEA-`), but allow customization. Prior to v1.5.0 the defaults were (`PBMM`) and (`PBMMAccel-`) respectively.
2. the Accelerator name and prefix ***CANNOT*** be changed after the initial installation;
3. the Accelerator prefix including the mandatory dash cannot be longer than 10 characters.
4. New installations, which now leverage Control Tower, require the `organization-admin-role` be set to `AWSControlTowerExecution` . Existing standalone installations will continue to utilize their existing role name for the `organization-admin-role` , typically `OrganizationAccountAccessRole` , as this role is used by AWS Organizations by default when no role name is specified while creating AWS accounts through the AWS console.
5. the Accelerator leverages this role name to create all new accounts in the organization;
6. this role name, as defined in the config file, ***MUST*** be utilized when manually creating all new sub-accounts in the Organization;
7. existing installs wishing to change the role name are required to first deploy a new role with a trust to the root account, in all accounts in the organization.

4.1.4 2.3. Accelerator Pre-Install Steps

2.3.1. General

Before installing, you must first:

1. Login to the Organization **Management (root) AWS account** with `AdministratorAccess`.
2. ***Set the region to your desired*** ***`home`*** ***region*** (i.e. `ca-central-1`)
3. Install AWS Control Tower:
4. Government of Canada customers are *required* to skip this step
5. OU and account names can ONLY be customized during initial installation. These values MUST match with the values supplied in the Accelerator config file.
6. Go to the AWS Control Tower console and click `Set up landing zone`
7. Select your `home` region (i.e. `ca-central-1`)
   - the Accelerator home region must match the Control Tower home region
8. Select *all* regions for `Additional AWS Regions for governance`, click `Next`
   - The Control Tower and Accelerator regions MUST be properly aligned
9. If a region is not `governed` by Control Tower, it must NOT be listed in `control-tower-supported-regions`
10. To manage a region requires the region:
    - be enabled in Control Tower (if supported)
    - added to the config file `control-tower-supported-regions` list (if supported)
    - added to the config file `supported-regions` list (even if not supported by Control Tower, as the Accelerator can manage regions not yet supported by Control Tower, but only when NOT listed in `control-tower-supported-regions`)
    - While we highly recommend guardrail deployment for all AWS enabled by default regions, at minimum
    - the home region MUST be enabled in Control Tower and must be listed in `control-tower-supported-regions`
    - both the home-region and ${GBL_REGION} must be listed in `supported-regions`
11. For the `Foundational OU`, leave the default value `Security`
12. For the `Additional OU` provide the value `Infrastructure`, click `Next`
13. Enter the email addresses for your `Log Archive` and `Audit` accounts, change the `Audit` account name to `Security`, click `Next`
    - OU and account names can ONLY be customized during initial installation. OU names, account names and email addresses *must* match identically with the values supplied in the Accelerator config file.
14. Click setup and wait ~60 minutes for the Control Tower installation to complete
15. Select `Add or register organizational units`, Click `Add an OU`
16. Type `Dev`, click `Add`, wait until the OU is finished provisioning (or it will error)
17. Repeat step 9 for each OU (i.e. `Test`, `Prod`, `Central`, `Sandbox`)
18. Select `Account factory`, Edit, Subnets: 0, Deselect all regions, click `Save`
19. In AWS Organizations, move the Management account from the `root` OU into the `Security` OU
20. Verify:
21. AWS Organizations is enabled in `All features` mode
    - if required, navigate to AWS Organizations, click `Create Organization`, `Create Organization`
22. Service Control Policies are enabled
    - if required, in Organizations, select `Policies`, `Service control policies`, `Enable service control policies`
23. Verify the Organization Management (root) account email address
24. In AWS Organizations, Settings, "Send Verification Request"
25. Once it arrives, complete the validation by clicking the validation link in the email
26. Create a new KMS key to encrypt your source configuration bucket (you can use an existing key)
27. AWS Key Management Service, Customer Managed Keys, Create Key, Symmetric, and then provide a key name (`ASEA-Source-Bucket-Key`), Next

28. Select a key administrator (Admin Role or Group for the Organization Management account), Next
29. Select key users (Admin Role or Group for the Organization Management account), Next
30. Validate an entry exists to "Enable IAM User Permissions" (critical step if using an existing key)

- `"arn:aws:iam::123456789012:root"`, where `123456789012` is your ***Organization Management*** account id.

31. Click Finish
32. Select the new key, Select `Key Rotation`, `Automatically rotate this CMK every year`, click Save.
33. Enable `"Cost Explorer"` (My Account, Cost Explorer, Enable Cost Explorer)
34. With recent platform changes, Cost Explorer *may* now be auto-enabled (unable to confirm)
35. Enable `"Receive Billing Alerts"` (My Account, Billing Preferences, Receive Billing Alerts)
36. It is ***extremely important*** that ***all*** the account contact details be validated in the Organization Management (root) account before deploying any new sub-accounts.
37. This information is copied to every new sub-account on creation.
38. Subsequent changes to this information require manually updating it in ***each*** sub-account.
39. Go to `My Account` and verify/update the information lists under both the `Contact Information` section and the `Alternate Contacts` section.
40. Please ESPECIALLY make sure the email addresses and Phone numbers are valid and regularly monitored. If we need to reach you due to suspicious account activity, billing issues, or other urgent problems with your account - this is the information that is used. It is CRITICAL it is kept accurate and up to date at all times.

2.3.2. Create GitHub Personal Access Token and Store in Secrets Manager

As of v1.5.0, the Accelerator offers deployment from either GitHub or CodeCommit:

GitHub (recommended)

1. You require a GitHub access token to access the code repository
2. Instructions on how to create a personal access token are located here.
3. Select the scope `public_repo` underneath the section `repo: Full control over private repositories`.
4. Store the personal access token in Secrets Manager as plain text. Name the secret `accelerator/github-token` (case sensitive).
5. Via AWS console

- Store a new secret, and select `Other type of secrets`, `Plaintext`
- Paste your secret with no formatting no leading or trailing spaces (i.e. completely remove the example text)
- Select the key you created above (`ASEA-Source-Bucket-Key`),
- Set the secret name to `accelerator/github-token` (case sensitive)
- Select `Disable rotation`

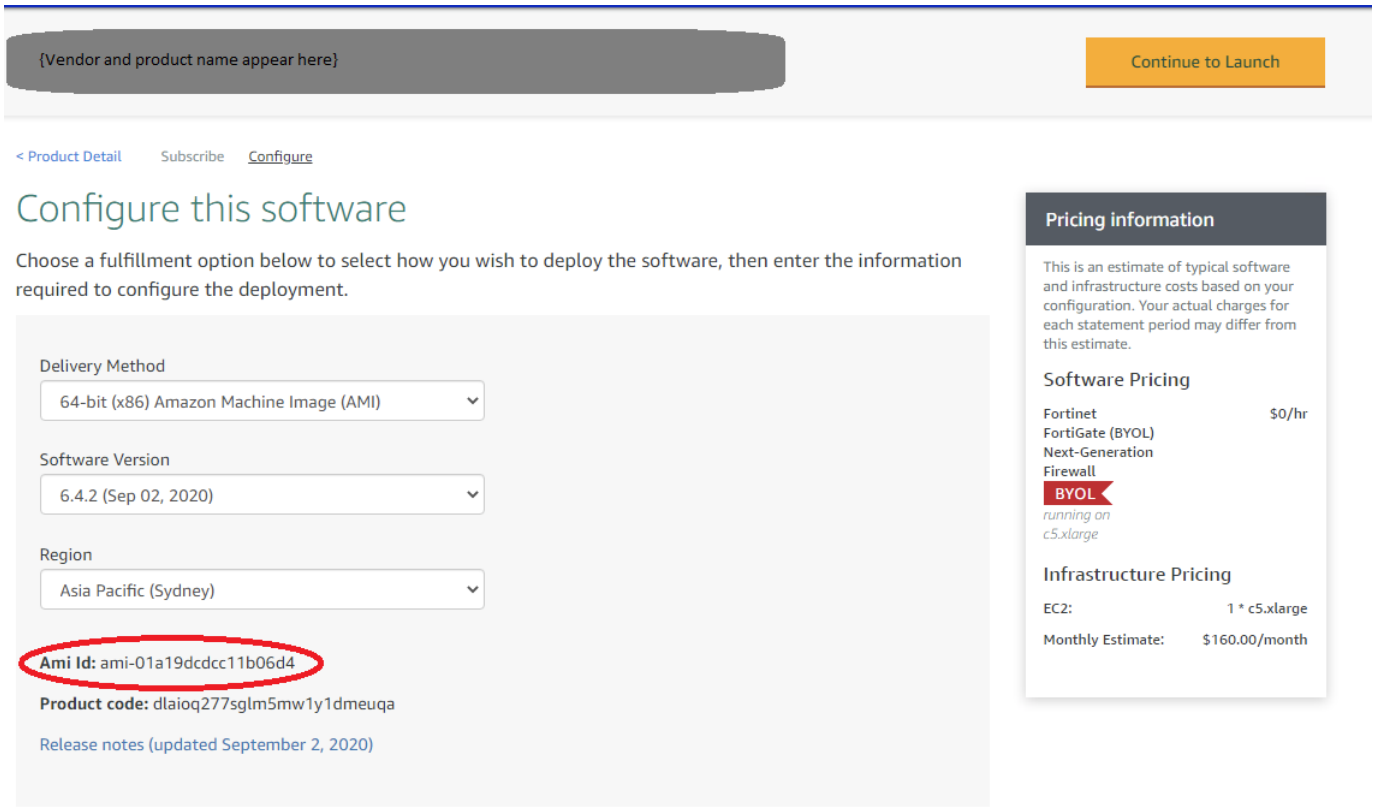
CodeCommit (alternative option)

- Multiple options exist for downloading the GitHub Accelerator codebase and pushing it into CodeCommit. As this option is only for advanced users, detailed instructions are not provided.
- In your AWS Organization Management account, open CodeCommit and create a new repository named `aws-secure-environment-accelerator`
- Go to GitHub and download the repository `Source code` zip or tarball for the release you wish to deploy
- Do NOT download the code off the main GitHub branch, this will leave you in a completely unsupported state (and with beta code)
- Push the extracted codebase into the newly created CodeCommit repository, maintaining the file/folder hierarchy
- Set the default CodeCommit branch for the new repository to main
- Create a branch following the Accelerator naming format for your release (i.e. `release/v1.5.0`)

2.3.3. AWS Internal (Employee) Accounts Only

If deploying to an internal AWS employee account and installing the solution with a 3rd party firewall, you need to enable Private Marketplace (PMP) before starting:

1. In the Organization Management account go here: https://aws.amazon.com/marketplace/privatemarketplace/create
2. Click `Create a Private Marketplace`, and wait for activation to complete
3. Go to the "Account Groups" sub-menu, click `Create account group`
4. Enter an Account Group Title (i.e. `Default`) and `Add` the Management (root) account number in `Associate AWS account`
5. Associate the default experience `New Private Marketplace`, then click `Create account group` and wait for it to create
6. Go to "Experiences" sub-menu, select `New Private Marketplace`
7. Select the "Settings" sub-tab, and click the `Not Live` slider to make it `Live` and wait for it to complete
8. Ensure the "Software requests" slider is set to `Requests off` and wait for it to complete
9. Change the name field (i.e. append `-PMP`) and change the color, so it is clear PMP is enabled for users, click `Update`
10. Go to the "Products" sub-tab, then select the `All AWS Marketplace products` nested sub-tab
11. Search Private Marketplace for the Fortinet or Checkpoint products and select
12. `Fortinet FortiGate (BYOL) Next-Generation Firewall` and
13. `Fortinet FortiManager (BYOL) Centralized Security Management` **or**
14. `CloudGuard Network Security for Gateway Load Balancer - BYOL` and
15. `Check Point Security Management (BYOL)`
16. Select "Add" in the top right
17. Due to PMP provisioning delays, this sometimes fails when attempted immediately following enablement of PMP or if adding each product individually - retry after 20 minutes.
18. While not used in this account, you must now subscribe to the two subscriptions and accept the EULA for each product (you will need to do the same in the perimeter account, once provisioned below)
    - To subscribe, select the "Approved products" tab
    - Click on the product you want to subscribe, in this case `Fortinet FortiGate (BYOL) Next-Generation Firewall` and `Fortinet FortiManager (BYOL Centralized Security Management` **or** `CloudGuard Network Security for Gateway Load Balancer - BYOL` and `Check Point Security Management (BYOL)`
    - Click on "Continue to Subscribe"
    - Click on "Accept Terms" and wait for subscription to be completed
    - If you are deploying in any region except ca-central-1 or wish to switch to a different license type, you need the new AMI id's. After successfully subscribing, continue one more step and click the “Continue to Configuration”. When you get the below screen, select your region and version (**Fortinet** `v6.4.7`, **Checkpoint Mgmt** `R81.10-335.883` and **CloudGuard** `R80.40-294.374` recommended at this time). Marketplace will provide the required AMI id. Document the two AMI id's, as you will need to update them in your config.json file below.



4.1.5 2.4. Basic Accelerator Configuration

1. Select a sample config file as a baseline starting point
2. **IMPORTANT: Use a config file from the Github code branch you are deploying from, as valid parameters change over time. The `main` branch is NOT the current release and often will not work with the GA releases.**
3. sample config files can be found in this folder;
4. descriptions of the sample config files and customization guidance can be found here;
5. unsure where to start, use the `config.lite-CTNFW-example.json`, where CTNFW is for Control Tower w/NFW;
6. These configuration files can be used, as-is, with only minor modification to successfully deploy the sample architectures;
7. On upgrades, compare your deployed configuration file with the latest branch configuration file for any new or changed parameters;
8. At minimum, you MUST update the AWS account names and email addresses in the sample file:
9. For existing accounts, they *must* match identically to both the account names and email addresses defined in AWS Organizations;
10. For new accounts, they must reflect the new account name/email you want created;
11. All new AWS accounts require a unique email address which has never before been used to create an AWS account;
12. When updating the budget or SNS notification email addresses within the sample config, a single email address for all is sufficient;
13. Update the IP address in the "alarm-not-ip" variable with your on-premise IP ranges (used for the AWS-SSO-Authentication-From-Unapproved-IP alarm);
14. If deploying the Managed AD, update the dns-domain, netbios-domain, log-group-name, as well as the AD users and groups that will be created;
15. For a test deployment, the remainder of the values can be used as-is;
16. While it is generally supported, we recommend not adding more than 1 or 2 workload accounts to the config file during the initial deployment as it will increase risks of hitting a limit. Once the Accelerator is successfully deployed, add the additional accounts to the config file and rerun the state machine.
17. More information here on the fields in the config file that need to be updated.

18. A successful deployment of the prescriptive architecture requires VPC access to 9 AWS endpoints, you cannot remove both the perimeter firewalls (all public endpoints) and the 9 required central VPC endpoints from the config file (ec2, ec2messages, ssm, ssmmessages, cloudformation, secretsmanager, kms, logs, monitoring).
19. When deploying to regions other than `ca-central-1`, you need to modify your config file as follows (for Canada Central 1, the AMI ids are pre-populated for you):
20. Update the firewall and firewall manager AMI id's to reflect your home regions regional AMI id's (see 2.3.3, item 13), making sure you select the right version and region per the recommendations.
21. Validate all the Interface Endpoints defined in your config file are supported in your home region (i.e. Endpoint VPC). Remove unsupported endpoints from the config file, add additional endpoints as available.
22. If you are installing into a home region which is explicitly named in any of the replacements\addl_regions_x, remove it from the list. If deploying in us-east-1, remove ${GBL_REGION}.
23. Create an S3 bucket in your Organization Management account `your-bucket-name`
24. you must supply this bucket name in the CFN parameters *and* in the config file (`global-options\central-bucket`)
25. the bucket name *must* be the same in both spots
26. the bucket must have versioning enabled
27. the bucket must be `S3-KMS` encrypted using the `ASEA-Source-Bucket-Key` created above
28. Place your customized config file(s), named `config.json` (or `config.yaml`), in your new bucket
29. If required, place the firewall configuration and license files in the folder and path defined in the config file
30. For AWS Network Firewall: `nfw/nfw-example-policy.json`
31. For Fortinet: `firewall/firewall-example.txt` , `firewall/license1.lic` and `firewall/license2.lic`

    - We have made several samples available here: `./reference-artifacts/Third-Party/`
    - Both samples comprise an active / active firewall pair. Until recently we only brought up one tunnel per firewall, you now also have an example which brings up both tunnels per firewall
    - If you updated your perimeter VPC subnet names, you must also make these changes in your firewall-example.txt file
    - If you don't have any license files, update the config file with an empty array (`"license": []`). Do NOT use the following: `[""]` .

32. The basic Checkpoint configuration is stored directly in config.json
33. Place any defined certificate files in the folder and path defined in the config file
34. i.e. `certs/example1-cert.key` , `certs/example1-cert.crt`
35. Sample available here: `./reference-artifacts/Certs-Sample/*`
36. Ideally you would generate real certificates using your existing certificate authority
37. Should you wish, instructions are provided to aid in generating your own self-signed certificates (Self signed certificates are NOT secure and simply for demo purposes)
38. Use the examples to demonstrate Accelerator TLS functionality only
39. Detach ***ALL*** SCPs (except `FullAWSAccess` which remains in place) from all OU's and accounts before proceeding
40. For Control Tower based installs do NOT detach Control Tower SCPs (i.e. aws-guardrails-xxxxxx)
41. Installation **will fail** if this step is skipped

4.1.6 2.5. Installation

1. You can find the latest release in the repository here.
2. We only support new installations of v1.3.9 or above (older releases continue to function)
3. Download the CloudFormation (CFN) template for the release you plan to install (either `AcceleratorInstallerXXX.template.json` for GitHub or `AcceleratorInstallerXXX-CodeCommit.template.json` for CodeCommit)
4. Use the provided CloudFormation template to deploy a new stack in your Management (root) AWS account
5. As previously stated we do not support installation in sub-accounts

6. Login to your Organization Management account and ***make sure you are in your desired `home` region*** (i.e. `ca-central-1`) (your desired primary or control region)
7. Navigate to **CloudFormation** in the AWS Console and click `Create stack with new resources (standard)`, then
8. Select "Template is ready"
9. For the "Specify template" select "Upload a template file"
10. Select the `*.template.json` file you downloaded in step 2 above
11. Click Next
12. Fill out the required parameters - ***LEAVE THE DEFAULTS UNLESS SPECIFIED BELOW***
13. Specify `Stack Name` STARTING with `ASEA-` (case sensitive) suggest a suffix of `orgname` or `username`
14. Change `ConfigS3Bucket` to the name of the bucket you created above `your-bucket-name`
15. Add an `Email` address to be used for State Machine Status notification
16. The `GithubBranch` should point to the release you selected
    - if upgrading, change it to point to the desired release
    - the latest stable branch is currently `release/v1.5.0`, case sensitive
    - click `Next`
17. Finish deploying the stack
18. Apply a tag on the stack, Key= `Accelerator`, Value= `ASEA` (case sensitive).
19. **ENABLE STACK TERMINATION PROTECTION** under `Stack creation options`
20. Click `Next`, Acknowledge resource creation, and click `Create stack`
21. The stack typically takes under 5 minutes to deploy.
22. Once deployed, you should see a CodePipeline project named `ASEA-InstallerPipeline` in your account. This pipeline connects to Github, pulls the code from the prescribed branch and deploys the Accelerator state machine.
23. if the CloudFormation fails to deploy with an `Internal Failure`, or, if the pipeline fails connecting to GitHub, then:
    - fix the issue with your GitHub secret created in section 2.3.2, then delete the Installer CloudFormation stack you just deployed, and restart at step 3 of this section.
24. For new stack deployments, when the stack deployment completes, the Accelerator state machine will automatically execute (in Code Pipeline). When upgrading you must manually `Release Change` to start the pipeline.
25. **While the pipeline is running:**
    - review the list of Known Installation Issues in section 2.5.1 below
    - review the Accelerator Basic Operation and Frequently Asked Questions (FAQ) Document
26. Once the pipeline completes (~10 mins), the main state machine, named `ASEA-MainStateMachine_sm`, will start in Step Functions
27. The state machine time is dependent on the quantity of resources being deployed. On an initial installation of a more complex sample configuration files, it takes approximately 2 hours to execute (depending on the configuration file). Timing for subsequent executions depends entirely on what resources are changed in the configuration file, but often takes as little as 20 minutes.
    - While you can watch the state machine in Step Functions, you will also be notified via email when the State Machine completes (or fails). Successful state machine executions include a list of all accounts which were successfully processed by the Accelerator.
28. The configuration file will be automatically moved into Code Commit (and deleted from S3). From this point forward, you must update your configuration file in CodeCommit.
29. You will receive an email from the State Machine SNS topic and the 3 SNS alerting topics. Please confirm all four (4) email subscriptions to enable receipt of state machine status and security alert messages. Until completed, you will not receive any email messages (must be completed within 7-days).

30. If the state machine **fails**:
    - Refer to the Troubleshooting Guide for instructions on how to inspect and retrieve the error
    - You can also refer to the FAQ and Known Installation Issues
    - Once the error is resolved, re-run the step function `ASEA-MainStateMachine_sm` using `{"scope": "FULL","mode": "APPLY"}` as input
31. If deploying a prescriptive architecture with 3rd party firewalls, after the perimeter account is created in AWS Organizations, but before the Accelerator reaches Stage 2:

a. NOTE: If you miss the step, or fail to execute it in time, no need to be concerned, you will simply need to re-run the main state machine (`ASEA-MainStateMachine_sm`) to deploy the firewall (no SM input parameters required)

b. Login to the **perimeter** sub-account (Assume your `organization-admin-role`)

c. Activate the 3rd party vendor firewall and firewall manager AMI's in the AWS Marketplace

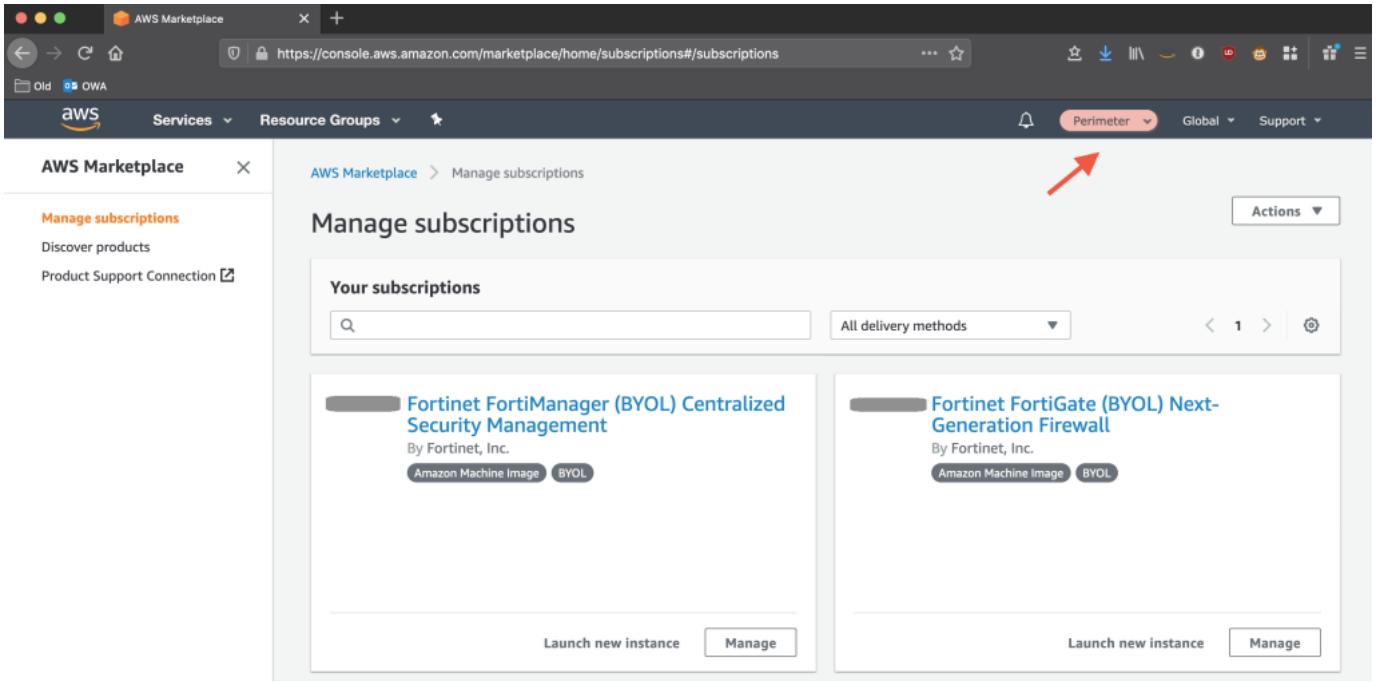
d. Navigate back to your private marketplace

e. Note: Employees should see the private marketplace, including the custom color specified in prerequisite step 4 above.

f. Select "Discover products" from the side bar, then in the "Refine Results" select "Private Marketplace => Approved Products"

g. Subscribe and Accept the Terms for each product (firewall and firewall manager)

h. When complete, you should see the marketplace products as subscriptions **in the Perimeter account**:



1. If deploying the prescriptive architecture, once the main state machine (`ASEA-MainStateMachine_sm`) completes successfully, confirm the status of your perimeter firewall deployment

- If you have t2.micro ec2 instances running in any account which had the account-warming flag set to true, they will be removed on the next state machine execution;
- If your perimeter firewalls were defined but not deployed on first run, you will need to rerun the state machine. This happens when:
- you were unable to activate the firewall AMI's before stage 2 (step 16)
- we were not able to fully activate your account before we were ready to deploy your firewalls. This case can be identified by a running EC2 micro instance in the account, or by looking for the following log entry 'Minimum 15 minutes of account warming required for account'.
- In these cases, simply select the `ASEA-MainStateMachine_sm` in Step Functions and select `Start Execution` (no SM input parameters required)

2.5.1. Known Installation Issues

Current Issues:

- When a new installation includes AWS Network Firewall (NFW), we are seeing State Machine failures in Phase 1 in the Perimeter account. Timing issues are causing the first deployment of the underlying CloudFormation stack to fail and rollback, when we automatically retry the stacks deployment, we attempt to recreate the NFW CloudWatch Log groups which were retained, causing a failure. A fix is in the works. Manually delete the two NFW log groups from the perimeter account (`/ASEA/Nfw/Central-Firewall/Alert` and `/ASEA/Nfw/Central-Firewall/Flow`) using either the Accelerator Pipeline Role or the Org Admin Role and rerun the state machine with the input of `{"scope": "FULL", "mode": "APPLY"}`.
- If dns-resolver-logging is enabled, VPC names containing spaces are not supported at this time as the VPC name is used as part of the log group name and spaces are not supported in log group names. By default in many of the sample config files, the VPC name is auto-generated from the OU name using a variable. In this situation, spaces are also not permitted in OU names (i.e. if any account in the OU has a VPC with resolver logging enabled and the VPC is using the OU as part of its name).
- On larger deployments we are occassionally seeing state machine failures when `Creating Config Recorders`. Simply rerun the state machine with the input of `{"scope": "FULL", "mode": "APPLY"}`.
- Occasionally CloudFormation fails to return a completion signal. After the credentials eventually fail (1 hr), the state machine fails. Simply rerun the state machine with the input of `{"scope": "FULL", "mode": "APPLY"}`.
- Applying new Control Tower Detective guardrails fails in v1.5.0. This is fixed in the next release.

Issues in Older Releases:

- New installs to releases prior to v1.3.9 are no longer supported.
- Upgrades to releases prior to v1.3.8 are no longer supported.

4.1.7 2.6. Post-Installation

The Accelerator installation is complete, but several manual steps remain:

1. Enable and configure AWS SSO in your `home` region (i.e. ca-central-1)
2. Login to the AWS Console using your Organization Management account
3. Navigate to AWS Single Sign-On, click `Enable SSO`
4. Set the SSO directory to AD ("Settings" => "Identity Source" => "Identity Source" => click `Change`, Select Active Directory, and select your domain from the list)
5. Under "Identity Source" section, Click `Edit` beside "Attribute mappings", then set the `email` attribute to: `${dir:email}` and click `Save Changes`
6. Configure Multi-factor authentication, we recommend the following minimum settings:
7. Every time they sign in (always-on)
8. Security key and built-in authenticators
9. Authenticator apps
10. Require them to provide a one-time password sent by email to sign in
11. Users can add and manage their own MFA devices
12. Create all the default permission sets and any desired custom permission sets
13. e.g. Select `AWS accounts` from the side bar, select "Permission sets" tab then `Create permission set`
    - `Use an existing job function policy` => Next
    - Select job function policy `AdministratorAccess`
    - Add Tags, Review and Create
    - repeat for each default permission set and any required custom permission sets
14. For Control Tower based installations, remove the orphaned Permission Sets from each AWS accounts (select the account, expand Permission Sets, click Remove for each)
15. Map MAD groups to permission sets and accounts
16. Select `AWS accounts` from the side bar and select `AWS organization` tab
17. Select the accounts you want to map to each MAD group and click `Assign users`
18. Select your DNS domain e.g. `example.local`, and search for the group you would like to assign (e.g. `aws-` for the pre-created groups) and click `Search connected directory`
19. Select the desired group `aws-log-archive-View`
20. Select the permission set you would like to assign to the MAD group to (e.g. `ViewOnlyAccess`)
21. Click `Finish` (Note: if it fails during provisioning, simply select the failed accounts and click on "Retry changes")
22. AWS SSO should be the primary mechanism for all access to all AWS accounts in your Organization, to determine or update the login page for your organization:
23. Click on `Dashboard` within the AWS SSO console and note the `User portal URL`
24. Share this url with all your users
25. NOTE: the url prefix can only be changed ONCE (ever) using the settings tab, so update with caution
26. Any pre-created AWS MAD users passwords are available in secrets manager in the AWS management account. To reset these passwords login to the Operations account through AWS SSO, then:
27. Navigate to "Directory Service" , select the directory => Actions => Reset user password
28. Users can change their passwords from any MAD domain connected instance
29. Configure the new alb-forwarding feature (added in v1.5.0)
30. AWS ALB's are published using DNS names which resolve to backing IP's which could silently change at any time due to a scaling event, maintenance, or a hardware failure. While published as a DNS name, ALB's can only target IP addresses. This presents a challenge as we need the ALB's in the perimeter account to target ALB's in the various back-end workload accounts;
31. Alb-forwarding solves this problem by executing a small snippet of code every 60 seconds which updates managed alb listeners with any IP changes, ensuring any managed flows do not go offline. This removes the requirement to leverage a 3rd party appliance to perform NAT to a DNS name;

32. Alb-forwarding is enabled on the perimeter vpc in the perimeter account in many of the sample configs and can be used with Accelerator created or customer created ALB's. This setting causes a new DynamoDB table named `ASEA-Alb-Ip-Forwarding-vpc-*` to be created in the perimeter account. Additionally the parameter store in the perimeter account contains the details of all Accelerator deployed alb's across the environment for easy reference;
33. Stepts to configure:

- First you need to manually create a listener on the front-end alb (without a target group), multiple listeners are supported;
- Next, for each application that needs to be published, a record needs to be added to the DynamoDB table, see sample below;
- Records can be added to the table for any alb in the account running the alb-forwarding tool. Records can be added at any time. DDB change logs will trigger the initial creation of the appropriate target group(s) and IP addresses will be verified and updated every 60 seconds therafter.

Sample DynamoDB JSON to add an entry to the table:

```
{
    "id": "App1",
    "targetAlbDnsName": "internal-Core-mydevacct1-alb-123456789.ca-central-1.elb.amazonaws.com",
    "targetGroupDestinationPort": 443,
    "targetGroupProtocol": "HTTPS",
    "vpcId": "vpc-0a6f44a80514daaaf",
    "rule": {
      "sourceListenerArn": "arn:aws:elasticloadbalancing:ca-central-1:123456789012:listener/app/Public-DevTest-perimeter-alb/b1b12e7a0c412bf3/
ef9b022a4fdd8bdf",
      "condition": {
        "paths": ["/img/*", "/myApp2"],
        "hosts": ["aws.amazon.com"],
        "priority": 30
      }
    }
}
```

- where `id` is any unique text, `targetAlbDnsName` is the DNS address for the backend alb for this application (found in parameter store), `vpcId` is the vpc id containing the front-end alb (in this account), `sourceListenerArn` is the arn of the listener of the front-end alb, `paths` and `hosts` are both optional, but one of the two must be supplied. Finally, `priority` must be unique and is used to order the listener rules. Priorities should be spaced at least 40 apart to allow for easy insertion of new applications and forwarder rules. - the provided `targetAlbDnsName` must resolve to addresses within a [supported](https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-target-groups.html) IP address space.

1. On a per role basis, you need to enable the CWL Account Selector in the Security and the Operations accounts, in each account:
2. Go to CloudWatch, Settings, Under `Cross-account cross-region` select `Configure`, Under `View cross-account cross-region` select `Edit`, choose `AWS Organization account selector`, click `Save changes`
3. Configure central Ingress/Egress firewalls, if deployed
4. Layer 3/4 `appliance` based inspection is an optional feature

General

- If deployed, login to any 3rd party firewalls and firewall manager appliances and update any default passwords;
- Tighten security groups on the 3rd party firewall instances (using the Accelerator configuration file), further limiting access to firewall management interfaces to a set of designated and controlled CIDR ranges;
- Update the firewall configuration per your organizations security requirements and best practices;
- Diagrams reflecting perimeter traffic flows when NFW and/or GWLB are used can be found here on slides 6 through 9.

AWS Network Firewall

- The AWS Network Firewall policies and rules deployed by the Accelerator, can only be updated using the Accelerator. Customers wishing to manage the AWS Network Firewall from the console GUI, must create a new policy with new rules created through the console and then manually associate this new policy to the Accelerator deployed Network Firewall. Customers can choose either option, but they cannot be mixed to ensures that Accelerator updates do not overwrite console based updates.

Fortinet

- Manually update firewall configuration to forward all logs to the Accelerator deployed NLB addresses fronting the rsyslog cluster
- login to each firewall, select `Log Settings`, check `Send logs to syslog`, put the NLB FQDN in the `IP Address/FQDN` field (stored in parameter store of perimeter account)
- Manually update the firewall configuration to connect perimeter ALB high port flows through to internal account ALB's
- Note: while this option is still available, a new alb-forwarding mechanism is available in v1.5.0 (see section 2 above) which simplifies and eliminates this more complicated "NAT to DNS name" option;
- login to each firewall, switch to `FG-traffic` vdom, select `Policies & Objects`, select `Addresses`, Expand `Addresses`
- Set `Prod1-ALB-FQDN` to point to a reliable sub-account ALB FQDN, this is used for full-path health checks on ***all*** ALB's
- Set additional `DevX-ALB-FQDN`, `TestX-ALB-FQDN` and `ProdX-ALB-FQDN` to point to workload account ALB FQDNs
- Two of each type of ALB FQDN records have been created, when you need more, you need to create BOTH an additional FQDN and a new VIP, per ALB
- Each new VIP will use a new high port (i.e. 7007, 7008, etc.), all of which map back to port 443
- Detailed steps can be read here.

Checkpoint

- Download and install Checkpoint SmartConsole client on your PC (only available for Windows, if you have a Mac, you can use Workspaces to install the client)
- Retrieve the firewall and firewall manager SSL certificate from secrets manager in the perimeter account and save to a pem file (convert to ppk on Windows)
- Login to the `AWS Console` for the `Perimeter` account
- Navigate to `Secrets Manager` and select `Secrets`
- Look for Secret name `ASEA/keypair/ASEA-Appliances` and click on it
- Go to the `Secret value` section and click on `Retrieve secret value`
- Make sure you have the `Plaintext` tab selected and copy the content to a .pem file using your preferred text editor. (the file will need to be converted to ppk format on Windows)
- Wait approx. 25 min. after the managers "Launch time" and then SSH into the Firewall Manager using the SSL certificate (.pem file retrieved above) and login user `admin`
- The firewall manager EC2 instance name is `ASEA-Checkpoint-FirewallMgr`
- Once you SSH successfully, execute the following commands:
- `set user admin password`
- `set expert-password`
- `set user admin shell /bin/bash`
- `save config`
- The following commands are useful for troubleshooting (in expert mode):
- `autoprov_cfg -v` (check cme at Take 155 or greater)
- `autoprov_cfg show all` (check cme configuration)
- `cat /var/log/aws-user-data.log` (validate bootstrap, file should end with `"Publish operation" succeeded (100%)`)
- `tail -f /var/log/CPcme/cme.log` (watch to ensure it finds the instances, establishes SIC and adds the nodes)
- Login to SmartConsole, and update the firewall policy per your organizations security requirements
- An outbound rule allowing http and https should exist
- From the RDGW host in Operations, test to see if outbound web browsing is enabled
- NOTES:
- No best practice or security configuration has been configured on the Checkpoint firewalls. These firewalls have been configured to work with GWLB, but otherwise have the default/basic Checkpoint out-of-box configuration installed
- Do NOT reboot the Checkpoint appliances until bootstrap is complete (~25 minutes for the manager), or you will be required to redeploy the instance
- Recover root passwords for all sub-accounts and apply strong passwords
- Process documented here
- Enable MFA for **all** IAM users and **all** root account users, recommendations:
- Yubikeys provide the strongest form of MFA protection and are strongly encouraged for all account root users and all IAM users in the Organization Management (root) account
- the Organization Management (root) account requires a dedicated Yubikey (if access is required to a sub-account root user, we do not want to expose the Organization Management accounts Yubikey)
- every ~50 sub-accounts requires a dedicated Yubikey (minimize the required number of Yubikeys and the scope of impact should a Yubikey be lost or compromised)
- each IAM breakglass user requires a dedicated Yubikey, as do any additional IAM users in the Organization Management (root) account. While some CSPs do not recommend MFA on the breakglass users, it is strongly encouraged in AWS

- all other AWS users (AWS SSO, IAM in sub-accounts, etc.) can leverage virtual MFA devices (like Google Authenticator on a mobile device)
- Customers are responsible for the ongoing management and rotation of all passwords on a regular basis per their organizational password policy. This includes the passwords of all IAM users, MAD users, firewall users, or other users, whether deployed by the Accelerator or not. We do NOT automatically rotate any passwords, but strongly encourage customers do so, on a regular basis.
- During the installation we request required limit increases, resources dependent on these limits will not be deployed
- Limit increase requests are controlled through the Accelerator configuration file `"limits":{}` setting
- The sample configuration file requests increases to your EIP count in the perimeter account and to the VPC count and Interface Endpoint count in the shared-network account
- You should receive emails from support confirming the limit increases
- On the next state machine execution, resources blocked by limits should be deployed (i.e. additional VPC's and Endpoints)
- If more than 2 days elapses without the limits being increased, on the next state machine execution, they will be re-requested
- Note: After a successful install the Control Tower `Organizational units` dashboard will indicate `2 of 3` in the `Accounts enrolled` column for the Security OU, as it does not enable enrollment of the management account in guardrails. The Accelerator compliments Control Tower and enables guardrails in the management account which is important to high compliance customers.

4.2 3. Upgrades

4.2.1 3.1. Considerations

- Due to some breaking dependency issues, customers can only upgrade to v1.3.8 or above (older releases continue to function, but cannot be installed).
- While an upgrade path is planned, customers with a standalone Accelerator installation can upgrade to v1.5.0 but need to continue with a standalone installation until the Control Tower upgrade option becomes available.
- Always compare your configuration file with the config file from the release you are upgrading to in order to validate new or changed parameters or changes in parameter types / formats.
- do NOT update to the latest firewall AMI - see the the last bullet in section 5.1. Accelerator Design Constraints / Decisions
- do NOT update the `organization-admin-role` - see bullet 2 in section 2.2.7. Other
- do NOT update account-keys (i.e. existing installations cannot change the internal values to `management` from `master`)
- do NOT make changes outside those required for the upgrade (those stated in the release notes or found through the comparison with the sample config file(s)). Customers wishing to change existing Accelerator configuration should either do so before their upgrade, ensuring a clean/ successful state machine execution, or after a successful upgrade.
- The Accelerator name and prefix ***CANNOT*** be changed after the initial installation
- Customers which customized any of the Accelerator provided default configuration files (SCPs, rsyslog config, ssm-documents, iam-policies, etc.) must manually merge the latest Accelerator provided updates with deployed customizations:
- it is important customers assess the new defaults and integrate them into their custom configuration, or Accelerator functionality could break or Accelerator deployed features may be unprotected from modification
- if customers don't take action, we continue to utilize the deployed customized files (without the latest updates)
- The below release specific considerations need to be cumulatively applied (an upgrade from v1.2.3 to v1.2.5 requires you to follow both v1.2.4 and v1.2.5 considerations)

Release Specific Upgrade Considerations:

- Upgrades to `v1.5.0`:
- Due to the size and complexity of this upgrade, we require all customers to upgrade to `v1.3.8 or above` before beginning this upgrade
- While v1.5.0 supports Control Tower for *NEW* installs, existing Accelerator customers *CANNOT* add Control Tower to their existing installations at this time (planned enhancement for 22H1)
- Attempts to install Control Tower on top of the Accelerator will corrupt your environment (both Control Tower and the Accelerator need minor enhancements to enable)
- **The v1.5.0 custom upgrade guide can be found here**
- Upgrades to `v1.3.9 and above` from `v1.3.8-b and below`:
- All interface endpoints containing a period must be removed from the config.json file either before or during the upgrade process
- i.e. ecr.dkr, ecr.api, transfer.server, sagemaker.api, sagemaker.runtime in the full config.json example
- If you remove them on a pre-upgrade State Machine execution, you can put them back during the upgrade, if you remove them during the upgrade, you can put them back post upgrade.
- Upgrades to `v1.3.3 and above` from `v1.3.2 and below`:
- Requires mandatory config file schema changes as documented in the release notes.
- These updates cause the config file change validation to fail and require running the state machine with the following input to override the validation checks on impacted fields: `{"scope": "FULL", "mode": "APPLY", "configOverrides": {"ov-ou-vpc": true, "ov-ou-subnet": true, "ov-acct-vpc": true }}`
- Tightens VPC interface endpoint security group permissions and enables customization. If you use VPC interface endpoints that requires ports/protocols other than TCP/443 (such as email-smtp), you must customize your config file as described here
- Upgrades to `v1.3.0 and above` from `v1.2.6 and below`:
- **Please note MAJOR changes to state machine behavior, as documented here.**
- Upgrades to `v1.2.6 and above` from `v1.2.5 and below` - Ensure you apply the config file changes described in the release notes:
- Cut-paste the new `"replacements": {},` section at the top of the example config file into your config file, as-is
- Enables customers to leverage the repo provided SCP's without customization, simplifying upgrades, while allowing SCP region customization
- the cloud-cidrX/cloud-maskX variables are examples of customer provided values that can be used to consistently auto-replace values throughout config files, these 4 specific variables are ***all*** required for the firewalls to successfully deploy
- The new ${variable} are auto-replaced across your config files, SCP's and firewall config files.
- as the variables should resolve to their existing values, you can leave your config file using hardcoded region and Accelerator prefix naming, or you can update them to make subsequent file comparisons easier for future upgrades. These are most useful for new installations in non ca-central-1 regions
- Some repo provide filenames have changed, where they are referenced within the config file, you must update them to their new filenames
- We do not delete/cleanup old/unused SCP's, in case they were also used by customers for unmanaged OUs or sub-ou's. After the upgrade, you should manually delete any old/extra SCP's which are no longer required
- Upgrades to `v1.2.5 and above` from `v1.2.4 and below` requires the manual removal of the `PBMMAccel-PipelineRole` StackSet before beginning your upgrade (we have eliminated all use of StackSets in this release)
- In the root AWS account, go to: CloudFormation, StackSets
- Find: `ASEA-PipelineRole`, and Select the: `Stack Instances` tab
- Document all the account numbers, comma separated i.e. 123456789012, 234567890123, 345678901234
- Select: Actions, Delete Stacks from StackSets
- Paste the above account numbers (comma separated) in the Account numbers box
- Select the Accelerator installation/home region from the Specify Regions Box (should be the only region in the dropdown)
- Change: Concurrency to: 8, Next, Submit
- Wait for operation to complete (refresh the browser several times)

- Select Actions, Delete StackSet, click Delete StackSet
- Wait for the operation to complete
- Upgrades to `v1.2.4 and above` from `v1.2.3 and below` - Ensure you apply the config file changes described in the release notes:
- failure to set `"central-endpoint": true` directly on the endpoint VPC (instead of in global-options), will result in the removal of your VPC endpoints
- failure to move your zone definitions to the endpoint VPC, will result in the removal of you Public and Private hosted zones

4.2.2 3.2. Summary of Upgrade Steps (all versions)

1. Login to your Organization Management (root) AWS account with administrative privileges
2. Either: a) Ensure a valid Github token is stored in secrets manager (section 2.3.2) b) Ensure the latest release is in a valid branch of CodeCommit in the Organization Management account
3. Review and implement any relevant tasks noted in the upgrade considerations in section 3.1
4. Update the config file in CodeCommit with new parameters and updated parameter types based on the version you are upgrading to (this is important as features are iterating rapidly)
5. An automated script is available to help convert config files to the new v1.5.0 format
6. Compare your running config file with the sample config file from the latest release
7. Review the `Config file changes` section of the release notes for **all** Accelerator versions since your current deployed release
8. If you customized any of the other Accelerator default config files by overriding them in your S3 input bucket, merge the latest defaults with your customizations before beginning your upgrade
9. Download the latest installer template (`AcceleratorInstallerXYZ.template.json` or `AcceleratorInstallerXXX-CodeCommit.template.json`) from the `Assets` section of the latest release
10. Do ***NOT*** accidentally select the `ASEA-InitialSetup` CloudFormation stack **below**
11. If you are replacing your GitHub Token:
12. Take note of the `AcceleratorName` , `AcceleratorPrefix` , `ConfigS3Bucket` and `NotificationEmail` values from the Parameters tab of your deployed Installer CloudFormation stack (`ASEA-what-you-provided`)
13. Delete the Installer CloudFormation stack (`ASEA-what-you-provided`)
14. Redeploy the Installer CloudFormation stack using the template downloaded in step 6, providing the values you just documented (changes to `AcceleratorName` or `AcceleratorPrefix` are not supported)
15. The pipeline will automatically run and trigger the upgraded state machine
16. If you are using a pre-existing GitHub token, or installing from CodeCommit:
17. Update the Installer CloudFormation stack using the template downloaded in step 5, updating the `GithubBranch` to the latest release (eg. `release/v1.5.0`)
18. Go to AWS CloudFormation and select the stack: `ASEA-what-you-provided`
19. Select Update, select Replace current template, Select Upload a template file
20. Select Choose File and select the template you downloaded in step 6 (`AcceleratorInstallerXYZ.template.json` or `AcceleratorInstallerXXX-CodeCommit.template.json`)
21. Select Next, Update `GithubBranch` parameter to `release/vX.Y.Z` where X.Y.Z represents the latest release
22. Click Next, Next, I acknowledge, Update
23. Wait for the CloudFormation stack to update (`Update_Complete` status) (Requires manual refresh)
24. Go To Code Pipeline and Release the ASEA-InstallerPipeline

4.3 4. Existing Organizations / Accounts

4.3.1 4.1. Considerations: Importing existing AWS Accounts / Deploying Into Existing AWS Organizations

- The Accelerator *can* be installed into existing AWS Organizations
- our early adopters have all successfully deployed into existing organizations
- Existing AWS accounts *can* also be imported into an Accelerator managed Organization
- Caveats:
- Per AWS Best Practices, the Accelerator deletes the default VPC's in all AWS accounts, worldwide. The inability to delete default VPC's in pre-existing accounts will fail the installation/account import process. Ensure default VPC's can or are deleted before importing existing accounts. On failure, either rectify the situation, or remove the account from Accelerator management and rerun the state machine
- The Accelerator will NOT alter existing (legacy) constructs (e.g. VPC's, EBS volumes, etc.). For imported and pre-existing accounts, objects the Accelerator prevents from being created using preventative guardrails will continue to exist and not conform to the prescriptive security guidance
- Existing workloads should be migrated to Accelerator managed VPC's and legacy VPC's deleted to gain the full governance benefits of the Accelerator (centralized flow logging, centralized ingress/egress, no IGW's, Session Manager access, existing non-encrypted EBS volumes, etc.)
- Existing AWS services will be reconfigured as defined in the Accelerator configuration file (overwriting existing settings)
- We do NOT support *any* workloads running or users operating in the Organization Management (root) AWS account. The Organization Management (root) AWS account MUST be tightly controlled
- Importing existing *workload* accounts is fully supported, we do NOT support, recommend and strongly discourage importing mandatory accounts, unless they were clean/empty accounts. Mandatory accounts are critical to ensuring governance across the entire solution
- We've tried to ensure all customer deployments are smooth. Given the breadth and depth of the AWS service offerings and the flexibility in the available deployment options, there may be scenarios that cause deployments into existing Organizations to initially fail. In these situations, simply rectify the conflict and re-run the state machine.
- If the Firewall Manager administrative account is already set for your organization, it needs to be unset before starting a deployment.

4.3.2 4.2. Process to import existing AWS accounts into an Accelerator managed Organization

- Newly invited AWS accounts in an Organization will land in the root ou
- Unlike newly created AWS accounts which immediately have a Deny-All SCP applied, imported accounts are not locked down as we do not want to break existing workloads (these account are already running without Accelerator guardrails)
- In AWS Organizations, select ALL the newly invited AWS accounts and move them all (preferably at once) to the correct destination OU (assuming the same OU for all accounts)
- In case you need to move accounts to multiple OU's we have added a 2 minute delay before triggering the State Machine
- Any accounts moved after the 2 minute window will NOT be properly ingested, and will need to be ingested on a subsequent State Machine Execution
- This will first trigger an automated update to the config file and then trigger the state machine after a 2 minute delay, automatically importing the moved accounts into the Accelerator per the destination OU configuration
- As previously documented, accounts CANNOT be moved between OU's to maintain compliance, so select the proper top-level OU with care
- If you need to customize each of the accounts configurations, you can manually update the configuration file either before or after you move the account to the correct ou
- if before, you also need to include the standard 4 account config file parameters, if after, you can simply add your new custom parameters to the account entry the Accelerator creates
- if you add your imported accounts to the config file, moving the first account to the correct ou will trigger the state machine after a 2 minutes delay. If you don't move all accounts to their correct ou's within 2 minutes, your state machine will fail. Simply finish moving all accounts to their correct OU's and then rerun the state machine.
- If additional accounts are moved into OUs while the state machine is executing, they will not trigger another state machine execution, those accounts will only be ingested on the next execution of the state machine

- customers can either manually initiate the state machine once the current execution completes, or, the currently running state machine can be stopped and restarted to capture all changes at once
- Are you unsure if an account had its guardrails applied? The message sent to the state machine Status SNS topic (and corresponding email address) on a successful state machine execution provides a list of all successfully processed accounts.
- The state machine is both highly parallel and highly resilient, stopping the state machine should not have any negative impact. Importing 1 or 10 accounts generally takes about the same amount of time for the Accelerator to process, so it may be worth stopping the current execution and rerunning to capture all changes in a single execution.
- We have added a 2 min delay before triggering the state machine, allowing customers to make multiple changes within a short timeframe and have them all captured automatically in the same state machine execution.

4.3.3 4.3. Deploying the Accelerator into an existing Organization

- As stated above, if the ALZ was previously deployed into the Organization, please work with your AWS account team to find the best mechanism to uninstall the ALZ solution
- Ensure all existing sub-accounts have the role name defined in `organization-admin-role` installed and set to trust the Organization Management (root) AWS Organization account
- prior to v1.2.5, this role must be named: `AWSCloudFormationStackSetExecutionRole`
- if using the default role (`AWSCloudFormationStackSetExecutionRole`) we have provided a CloudFormation stack which can be executed in each sub-account to simplify this process
- As stated above, we recommend starting with new AWS accounts for the mandatory functions (shared-network, perimeter, security, log-archive accounts).
- To better ensure a clean initial deployment, we also recommend the installation be completed while ignoring most of your existing AWS sub-accounts, importing them post installation:
- create a new OU (i.e. `Imported-Accounts`), placing most of the existing accounts into this OU temporarily, and adding this OU name to the `global-options\ignored-ous` config parameter;
- any remaining accounts must be in the correct ou, per the Accelerator config file;
- install the Accelerator;
- import the skipped accounts into the Accelerator using the above import process, paying attention to the below notes
- NOTES:
- Do NOT move any accounts from any `ignored-ous` to the root ou, they will immediately be quarantined with a Deny-All SCP, they need to be moved directly to their destination ou
- As stated above, when importing accounts, there may be situations we are not able to fully handle
- If doing a mass import, we suggest you take a quick look and if the solution is not immediately obvious, move the account which caused the failure back to ignored-ous and continue importing the remainder of your accounts. Once you have the majority imported, you can circle back and import outstanding problem accounts with the ability to focus on each individual issue
- The challenge could be as simple as someone has instances running in a default VPC, which may require some cleanup effort before we can import (coming soon, you will be able to exclude single account/region combinations from default VPC deletion to gain the benefits of the rest of the guardrails while you migrate workloads out of the default VPC)

4.4 5. Notes

4.4.1 5.1. Accelerator Design Constraints / Decisions

- The Organization Management (root) account does NOT have any preventative controls to protect the integrity of the Accelerator codebase, deployed objects or guardrails. Do not delete, modify, or change anything in the Organization Management (root) account unless you are certain as to what you are doing. More specifically, do NOT delete, or change *any* buckets in the Organization Management (root) account.
- While generally protected, do not delete/update/change s3 buckets with cdk-asea-, or asea- in *any* sub-accounts.
- ALB automated deployments only supports Forward and not redirect rules.

- The Accelerator deploys SNS topics to send email alerts and notifications. Given email is not a secure transport mechanism, we have chosen not to enable SNS encryption on these topics at this time.
- AWS generally discourages cross-account KMS key usage. As the Accelerator centralizes logs across an entire organization as a security best practice, this is an exception/example of a unique situation where cross-account KMS key access is required.
- The Accelerator aggregates all logs in the log-archive account using Kinesis Data and Kinesis Firehose as aggregation tools where the logs could persist for up to 24 hours. These logs are encrypted with Customer Managed KMS keys once stored in S3 (ELB logs only support AES256). These logs are also encrypted in transit using TLS encryption. At this time, we have not enabled Kinesis at-rest encryption, we will reconsider this decision based on customer feedback.
- AWS Config Aggregator is deployed in the Organization Management (root) account as enablement through Organizations is simpler to implement. AWS Organizations only supported deploying the Aggregator in the Organization Management (root) account and not in a designated administrative account when we implemented this feature. We have a backlog item to update the code to move the Aggregator to the security account.
- An Organization CloudTrail is deployed, which is created in the primary region in the Organization Management (root) AWS account. All AWS account CloudTrails are centralized into this single CloudWatch Log Group. Starting in v1.1.9 this is where we deploy the CloudWatch Alarms which trigger for ALL accounts in the organization. Security Hub will erroneously report that the only account and/or region that is compliant with certain rules is the primary region of the Organization Management (root) account. We are working with the Security Hub team to rectify this situation in future Security Hub/Accelerator releases (resolved in Accelerator v1.5.0).
- Only 1 auto-deployed MAD in any mandatory-account is supported today.
- VPC Endpoints have no Name tags applied as CloudFormation does not currently support tagging VPC Endpoints.
- If the Organization Management (root) account coincidentally already has an ADC with the same domain name, we do not create/deploy a new ADC. You must manually create a new ADC (it won't cause issues).
- 3rd party firewall updates are to be performed using the firewall OS based update capabilities. To update the AMI using the Accelerator, you must first remove the firewalls and then redeploy them (as the EIP's will block a parallel deployment), or deploy a second parallel FW cluster and de-provision the first cluster when ready.
- When adding more than 100 accounts to an OU which uses shared VPC's, you must *first* increase the Quota `Participant accounts per VPC` in the shared VPC owner account (i.e. shared-network). Trapping this quota before the SM fails has been added to the backlog.
- The default limit for Directory Sharing is 125 accounts for an Enterprise Managed Active Directory (MAD), a quota increase needs to be manually requested through support from the account containing the MAD before this limit is reached. Standard MAD has a sharing limit of 5 accounts (and only supports a small quota increase). The MAD sharing limit is not available in the Service Quota's tools.

...Return to Accelerator Table of Contents

5. Operations & Troubleshooting

5.1 1. Operations & Troubleshooting Guide

5.2 2. Purpose

This document is targeted at individuals installing or executing the AWS Secure Environment Accelerator. It is intended to guide individuals who are executing the Accelerator by providing an understanding as to what happens at each point throughout execution and to assist in troubleshooting state machine failures and/or errors. This is one component of the provided documenation package and should be read after the Installation Guide, but before the Developer Guide.

5.3 3. System Overview

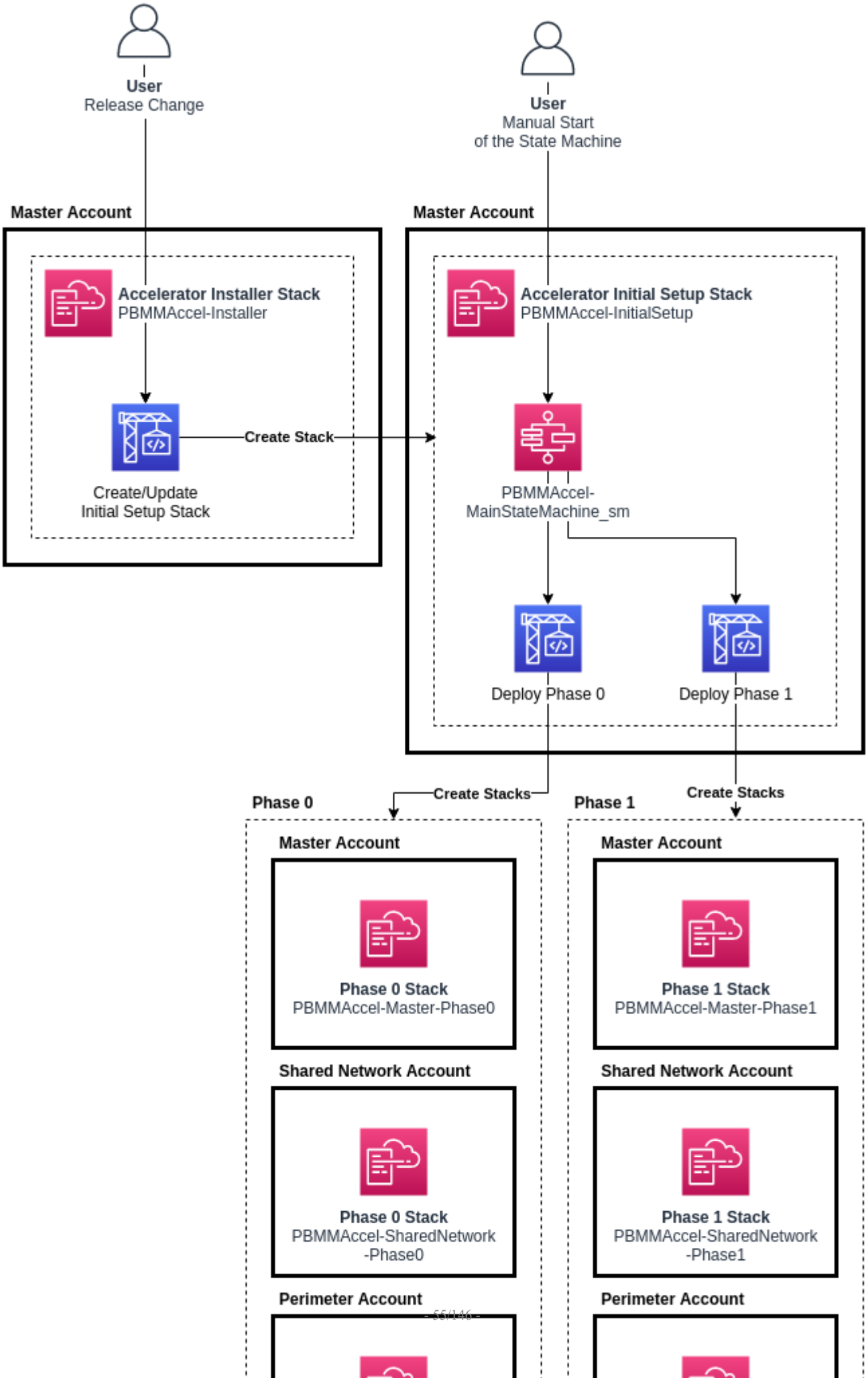
The system can be thought of in two levels. The first level of the system consists of Accelerator stacks and resources. Let's call these the Accelerator-management resource. The second level of the system consists of stacks and resources that are deployed by the Accelerator-management resource. Let's call these the Accelerator-managed resources. The Accelerator-management resources are responsible for deploying the Accelerator-managed resources.

There are two Accelerator-management stacks:

- the `Installer` stack that is responsible for creating the next listed stack;
- the `Initial Setup` stack. This stack is responsible for reading configuration file and creating Accelerator-managed resources in the relevant accounts.

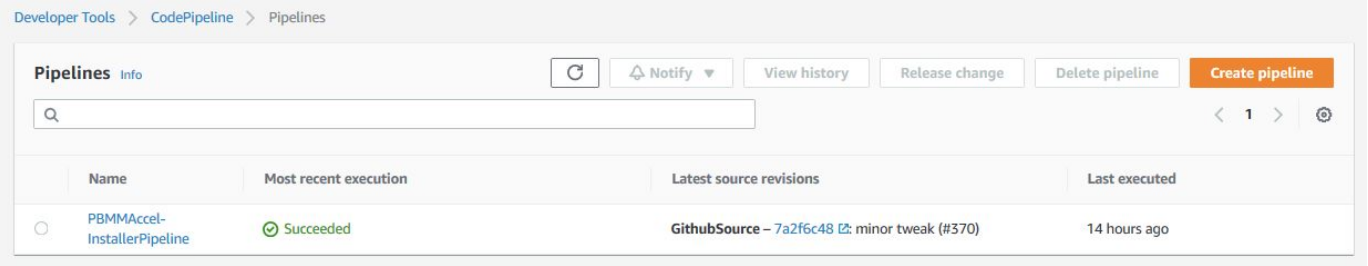
There are multiple Accelerator-managed stacks. Currently there are as many as twelve Accelerator-managed stacks per managed account.

The figure below shows a zoomed-out overview of the Accelerator. The top of the overview shows the Accelerator-management resources, i.e. the `Installer` stack and the `Initial Setup` stack. The bottom of the overview shows the Accelerator-managed resources in the different accounts.


User
Release Change
User
Manual Start
of the State Machine
Master Account
Accelerator Installer Stack
PBMMAccel-Installer
Create Stack
Create/Update
Initial Setup Stack
Master Account
Accelerator Initial Setup Stack
PBMMAccel-InitialSetup
PBMMAccel-
MainStateMachine_sm
Deploy Phase 0
Deploy Phase 1
Create Stacks
Create Stacks
Phase 0
Phase 1
Master Account
Phase 0 Stack
PBMMAccel-Master-Phase0
Shared Network Account
Phase 0 Stack
PBMMAccel-SharedNetwork
-Phase0
Perimeter Account
Master Account
Phase 1 Stack
PBMMAccel-Master-Phase1
Shared Network Account
Phase 1 Stack
PBMMAccel-SharedNetwork
-Phase1
Perimeter Account

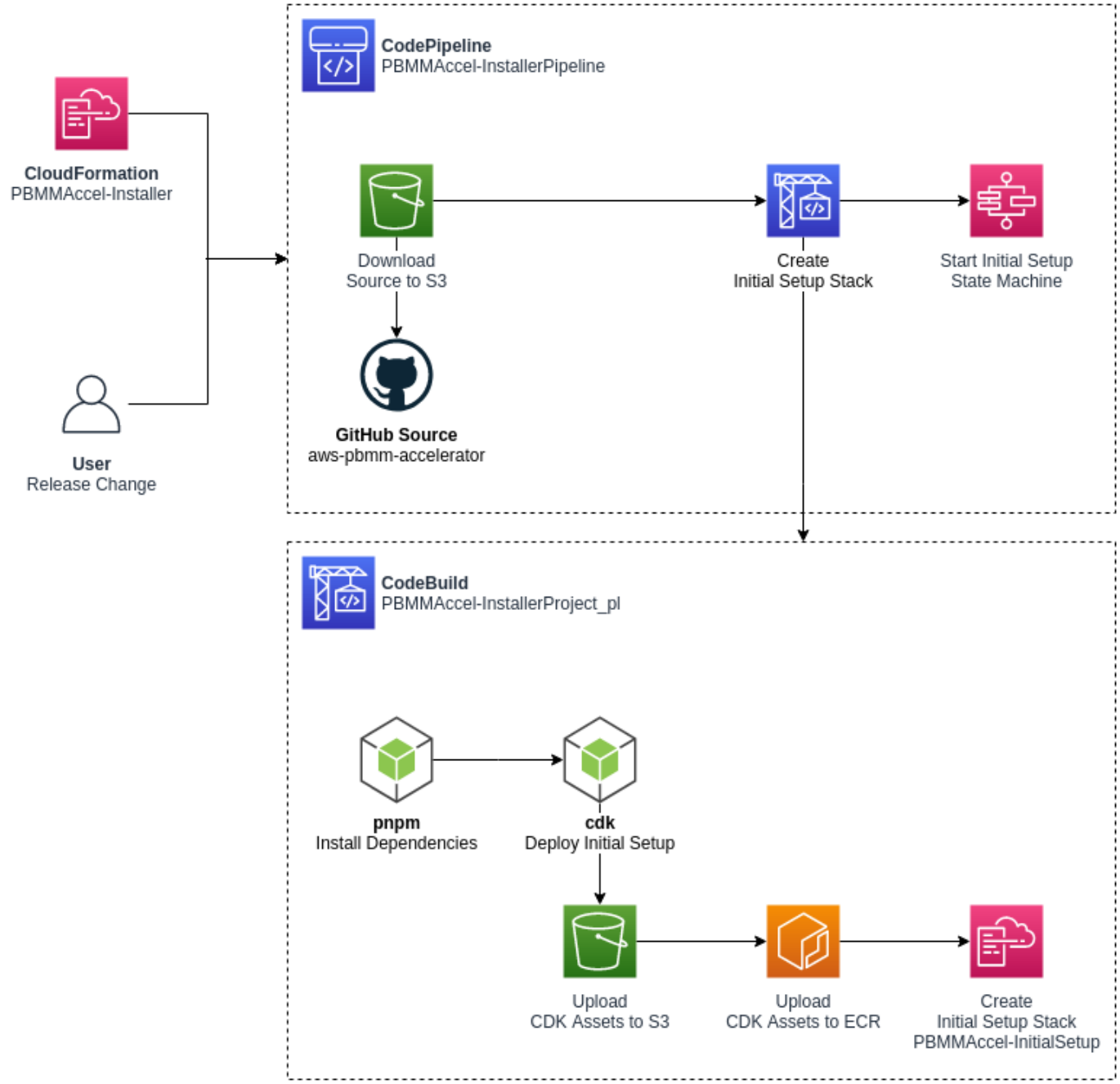
5.3.1 3.1. Installer Stack

The Accelerator-management `Installer` stack contains the necessary resources to deploy the Accelerator-management `Initial Setup` stack in an AWS account. This AWS account will be referred to as the 'root' account in this document.

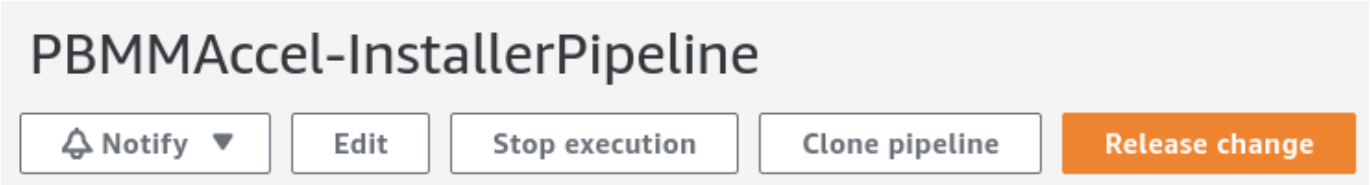


The Installer stack consists of the following resources:

- `ASEA-InstallerPipeline` : this is a `AWS::CodePipeline::Pipeline` that pulls the latest Accelerator code from GitHub. It launches the CodeBuild project `ASEA-InstallerProject_pl` , executes the `ASEA-Installer-SaveApplicationVersion` Lambda and launches the Accelerator state machine.
- `ASEA-InstallerProject_pl` : this is a `AWS::CodeBuild::Project` that installs the Accelerator in AWS account.
- `ASEA-Installer-SaveApplicationVersion` : this is a `AWS::Lambda::Function` that stores the current Accelerator version into Parameter Store.
- `ASEA-Installer-StartExecution` : this is a `AWS::Lambda::Function` that launches the Accelerator after CodeBuild deploys the Accelerator.
- Creation of AWS::DynamoDB::Table - `ASEA-Parameters` and `ASEA-Outputs` which are used for the internal operation of the Accelerator. `ASEA-Outputs` is used to share CloudFormation stack outputs between regions, stacks and phases. `ASEA-Parameters` is used to various configuration items like managed accounts, organizations structure, and limits.



The `ASEA-InstallerPipeline` starts when first installed using the CloudFormation template. The administrator can also start the pipeline manually by clicking the `Release Change` button in the AWS Console.



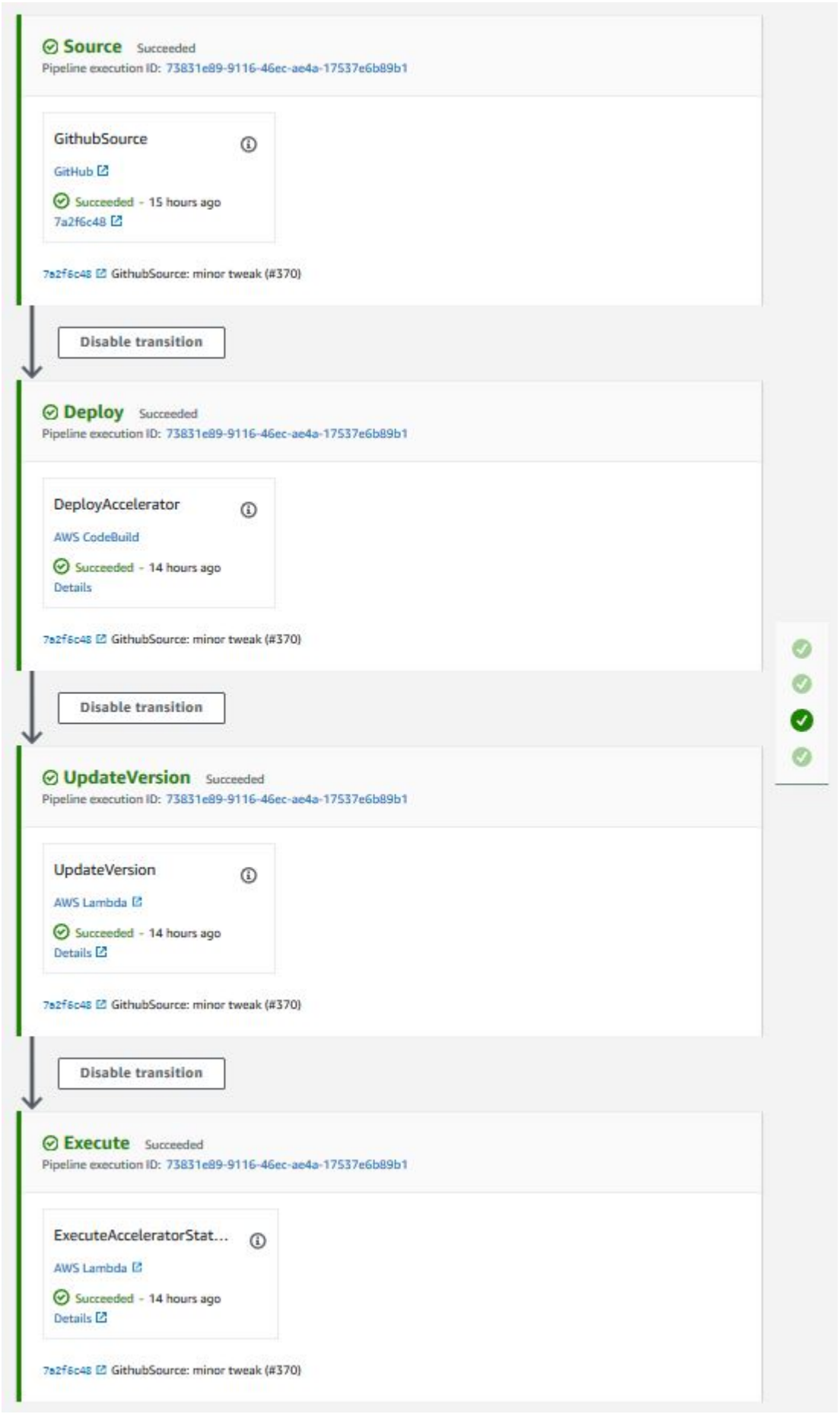
This starts the `ASEA-InstallerProject_pl` CodeBuild project. The CodeBuild project uses the GitHub source artifact. The CodeBuild projects spins up a new Linux instances and installs the Accelerator dependencies and starts the deployment of the Accelerator using the AWS Cloud Development Kit (CDK[1]).

CDK bootstraps its environment and creates the `CDKToolkit` stack in the AWS account. It creates the S3 bucket `cdktoolkit-stagingbucket-*` and the ECR repository `aws-cdk/assets`.

CDK copies assets to the bootstrap bucket and bootstrap repository that are used by the Accelerator. The assets that are stored on S3 include default IAM policies, default SCPs, default firewall configuration. The assets that are pushed to ECR include the Accelerator Docker build image. This Docker image is responsible for deploying Accelerator resources using the CDK.

CDK finally deploys the `Initial Setup` stack. The Accelerator state machine is described in the next section.

This diagram depicts the Accelerator Installer CodePipeline as of v1.2.1:

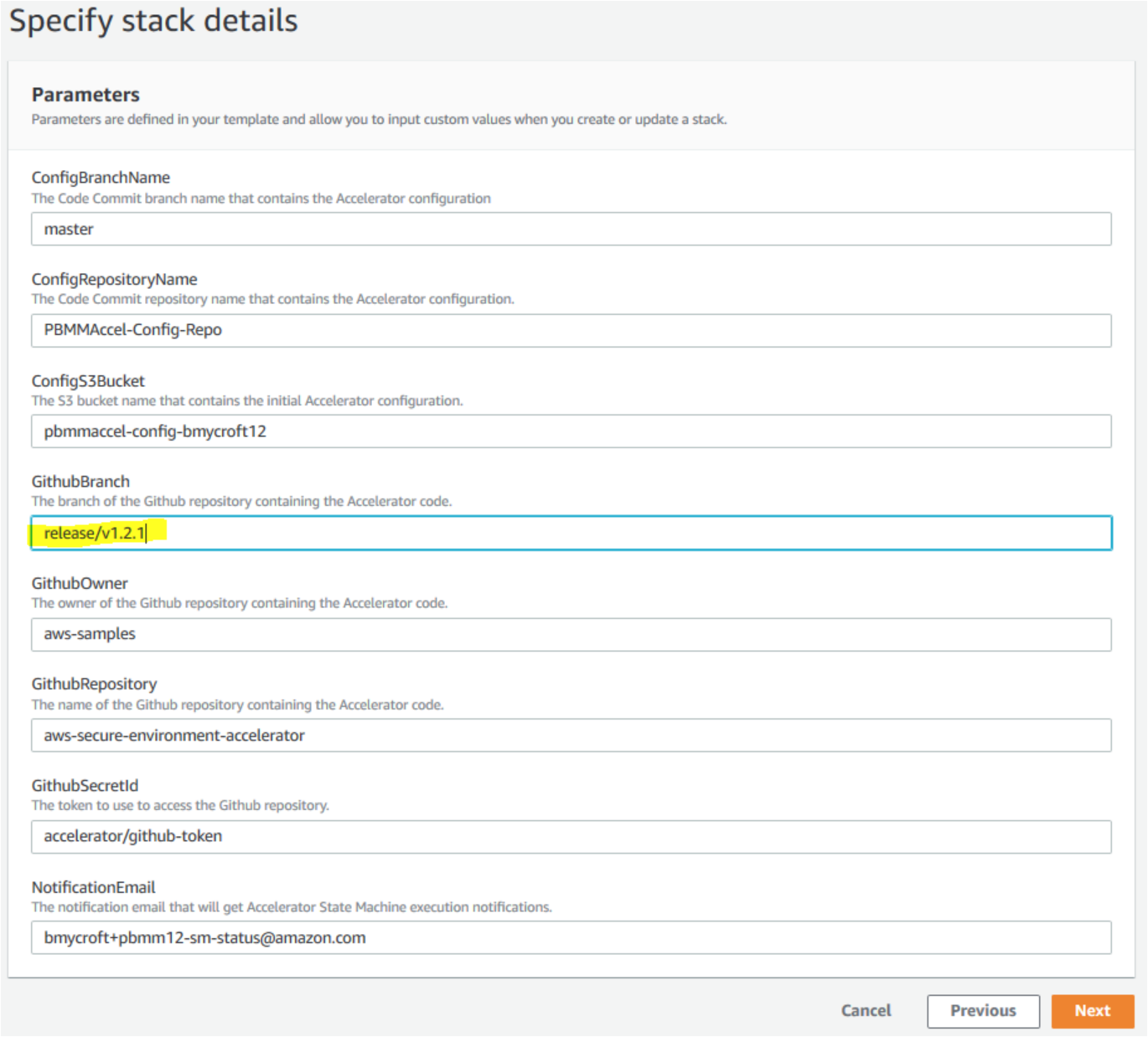

Source Succeeded
Pipeline execution ID: 73831e89-9116-46ec-ae4a-17537e6b89b1
GithubSource
GitHub
Succeeded - 15 hours ago
7a2f6c48
7a2f6c48 GithubSource: minor tweak (#370)
Disable transition
Deploy Succeeded
Pipeline execution ID: 73831e89-9116-46ec-ae4a-17537e6b89b1
DeployAccelerator
AWS CodeBuild
Succeeded - 14 hours ago
Details
7a2f6c48 GithubSource: minor tweak (#370)
Disable transition
UpdateVersion Succeeded
Pipeline execution ID: 73831e89-9116-46ec-ae4a-17537e6b89b1
UpdateVersion
AWS Lambda
Succeeded - 14 hours ago
Details
7a2f6c48 GithubSource: minor tweak (#370)
Disable transition
Execute Succeeded
Pipeline execution ID: 73831e89-9116-46ec-ae4a-17537e6b89b1
ExecuteAcceleratorStat...
AWS Lambda
Succeeded - 14 hours ago
Details
7a2f6c48 GithubSource: minor tweak (#370)

Once the Code Pipeline completes successfully:

- the Accelerator codebase was pulled from GitHub
- the Accelerator codebase was deployed/installed in the Organization Management (root) AWS account
- parameter store `/accelerator/version` was updated with the new version information
- this provides a full history of all Accelerator versions and upgrades
- the newly installed Accelerator state machine is started

At this time the resources deployed by the Installer Stack are no longer required. The Installer stack **could** be removed (which would remove the Code Pipeline) with no impact on Accelerator functionality.

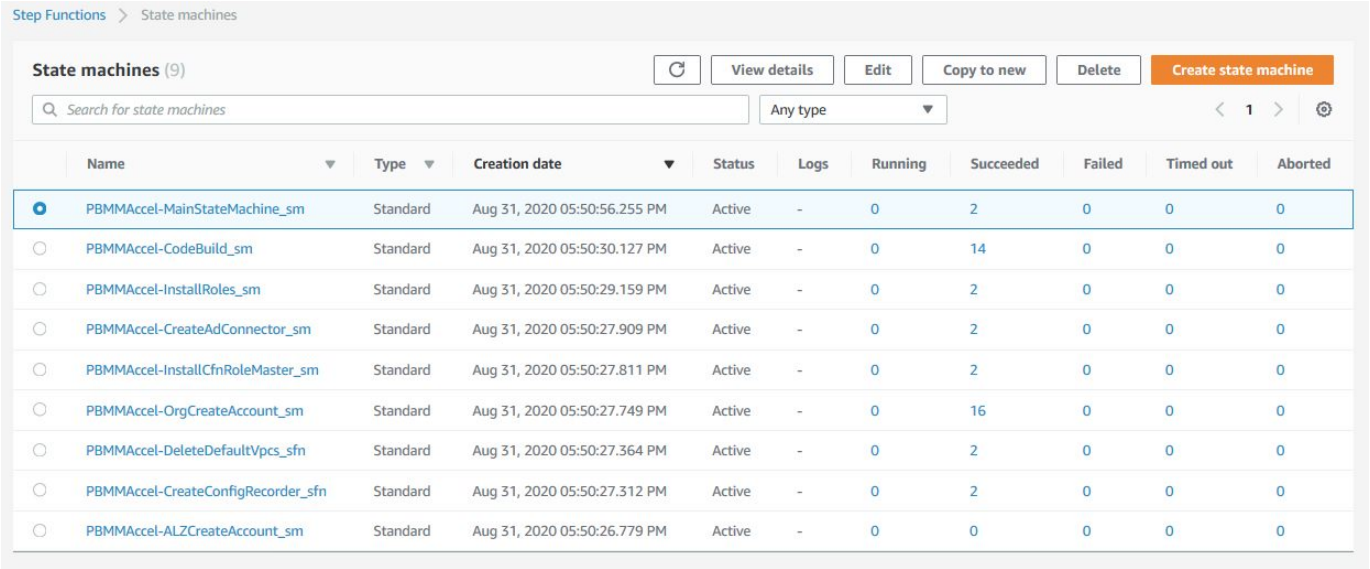
If the Installer Stack was removed, it would need to be re-installed to upgrade the Accelerator. If the stack was not removed, an Accelerator codebase upgrade often only requires updating a single stack parameter to point to the latest Accelerator code branch, and re-releasing the pipeline. No files to manually copy, change or update, an upgrade can be initiated with a simple variable update.



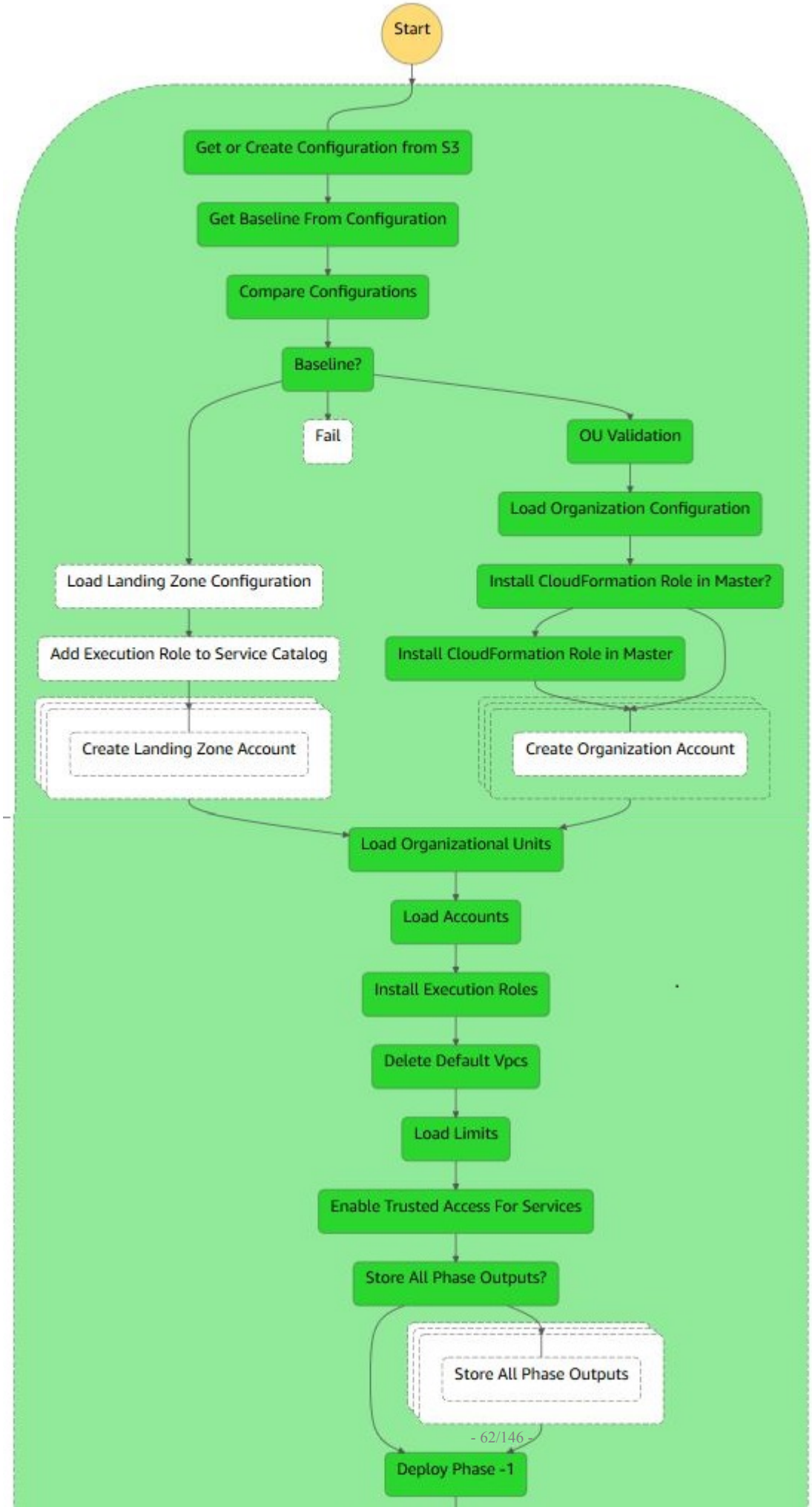
5.3.2 3.2. Initial Setup Stack

The Accelerator-management `Initial Setup` stack, named `ASEA-InitialSetup`, consists of a state machine, named `ASEA-MainStateMachine_sm`, that executes various steps to create the Accelerator-managed stacks and resources in the Accelerator-managed accounts. Using a state machine, we can clearly define the deployment process and systematically control branches of execution and handle exceptions.

The Accelerator comprises a primary state machine `ASEA-MainStateMachine_sm`, and nine supporting state machines (as of v1.2.1). Customer will only ever Execute the `ASEA-MainStateMachine_sm`. All troubleshooting will also typically begin with the `ASEA-MainStateMachine_sm`.



The image below depicts the latest state `ASEA-MainStateMachine_sm` machine. Each green or white square in the image represents a step in the state machine. This all green diagram represents a successul Accelerator state machine execution.


Start
Get or Create Configuration from S3
Get Baseline From Configuration
Compare Configurations
Baseline?
Fail
OU Validation
Load Organization Configuration
Load Landing Zone Configuration
Install CloudFormation Role in Master?
Add Execution Role to Service Catalog
Install CloudFormation Role in Master
Create Landing Zone Account
Create Organization Account
Load Organizational Units
Load Accounts
Install Execution Roles
Delete Default Vpcs
Load Limits
Enable Trusted Access For Services
Store All Phase Outputs?
Store All Phase Outputs
Deploy Phase -1

Notice the extremely linear state machine design. This was done to reduce complexity and ease troubleshooting. You may notice a small fork in the state machine. The left path is executed if the Accelerator is deployed on top of an ALZ, The right path is executed if the Accelerator is deployed as a standalone solution. If we eventually add a Control Tower deployment option, it is likely it could also leverage the existing ALZ path.

The state machine contains three different types of steps:

1. steps that execute a Lambda function;
2. steps that start another state machine, e.g. `Create Accounts` step;
3. steps that start another state machine that starts a CodeBuild project, e.g. the `Phase {-1,0,1,2,3,4,5}` steps.

The stack additionally consists of the following resources:

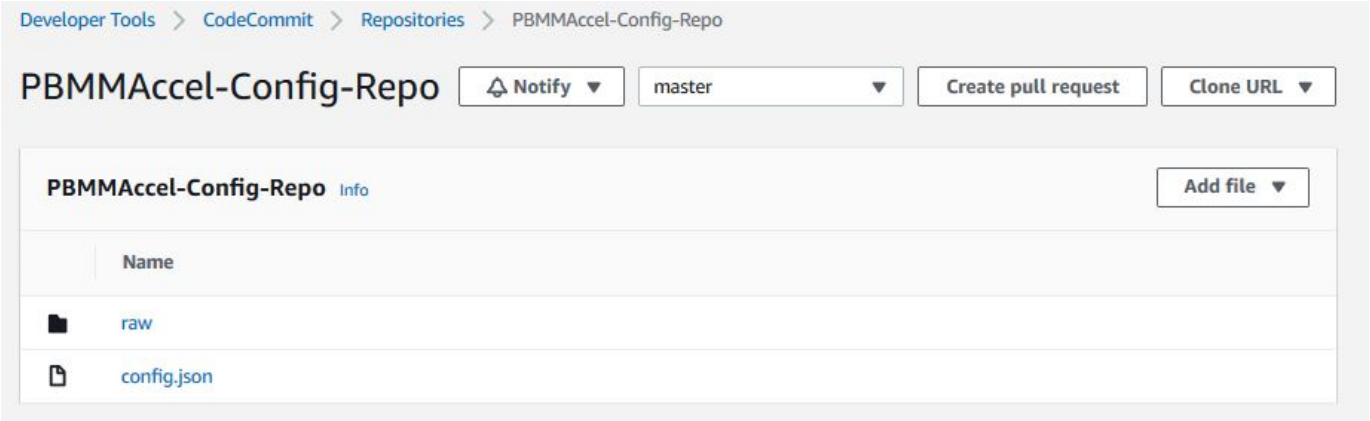
- AWS::CodeBuild::Project
- `ASEA-Deploy` or `ASEA-DeployPrebuilt`
- AWS::CodeCommit::Repository
- `ASEA-Config-Repo`
- AWS::IAM::Role
- `ASEA-L-SFN-MasterRole`
- `ASEA-L-SFN-Execution`
- AWS::Lambda::Function
- A Lambda function for every Lambda function step in the state machine.
- AWS::StepFunctions::StateMachine
- `ASEA-ALZCreateAccount_sm` : See *Create Landing Zone Account*;
- `ASEA-OrgCreateAccount_sm` : See *Create Organization Account*;
- `ASEA-InstallCfnRoleMaster_sm` : See Install CloudFormation Execution Role;
- `ASEA-InstallRoles_sm` : See *Install Execution Roles*;
- `ASEA-DeleteDefaultVpcs_sfn` : See *Delete Default VPCs*;
- `ASEA-CodeBuild_sm` : See *Deploy Phase 0*;
- `ASEA-CreateConfigRecorder_sfn` : See *Create Config Recorders*;
- `ASEA-CreateAdConnector_sm` : See *Create AD Connector*;
- `ASEA-StoreOutputs_sm` : See *Share Outputs* - new in v1.2.1.

Note: Most resources have a random suffix to their name. This is because we use CDK to deploy the resources. See https://docs.aws.amazon.com/cdk/latest/guide/identifiers.html#identifiers_logical_ids

3.2.1. Get or Create Configuration from S3

This step calls a Lambda function that finds or creates the configuration repository. Finds the configuration file(s) in the CodeCommit repository. If the configuration file cannot be found in the repository it is copied from the customer's S3 configuration bucket. If the copy is successful then the configuration file(s) in the S3 bucket will be removed.

The configuration file `config.json` or `config.yaml` is parsed and validated. This step will fail if both file types exist, the configuration file is not valid JSON or YAML or does not adhere to the configuration file specification. Internally the Accelerator always leverages JSON, but accepts JSON or YAML as the source input file and converts it to JSON prior to each execution, storing the converted and fully expanded file if in the raw folder.



3.2.2. Get Baseline from Configuration

This step calls a Lambda function that gets the `alz-baseline` of the configuration file to decide which path in the state machine will be taken.

3.2.3. Compare Configurations

This step calls a Lambda function that compares the previous version of the configuration file with the current version of the configuration file. The previous configuration file CodeCommit commit id is stored in the secret `accelerator/config/last-successful-commit` in AWS Secrets Manager in the root account.

The following configuration file changes are not allowed:

- changing ALZ baseline;
- changing root account or region;
- changing central log services account or region;
- changing the organizational unit, name or email address of an account;
- removing an account;
- changing the name, CIDR or region of a VPC;
- disabling a VPC;
- changing the name, availability zone, CIDR of a subnet;
- disabling or removing a subnet;
- changing the name, ASN, region or features of a transit gateway;
- changing the ID, VPC, subnet, region, size, DNS, Netbios of a Managed Active Directory;
- disabling a Managed Active Directory;
- changing the ASN of a virtual private gateway;
- changing the sharing to accounts of a VPC;
- changing the NACLs of a subnet.

It is possible to ignore certain configuration file changes. See Restart the State Machine how to pass these options to the state machine.

3.2.4. Load Landing Zone Configuration

Executed only when using AWS Landing Zone baseline

This step calls a Lambda function that validates the ALZ deployment and configuration.

This step fails when

- an existing ALZ deployment in the root account can not be found;
- an organizational unit that is used by ALZ or the Accelerator can not be found;
- an account is present in the Accelerator configuration but does not exist in AWS Organizations;
- an account has a name, email address or organizational unit that is different from the ALZ configuration;
- an account has a name, email address or organizational unit that is different from the Accelerator configuration;
- the organizational units in the ALZ configuration are different from the ones in AWS Organizations;
- the account type of ALZ configuration accounts can not be detected;
- the accounts of type `primary`', `security`, `log-archive`, `shared-services` are missing from the ALZ configuration.

3.2.5. Add Execution Role to Service Catalog

Executed only when using AWS Landing Zone baseline

This step calls a Lambda function that adds the state machine's IAM role to the ALZ Account Vending Machine (AVM) service catalog portfolio.

3.2.6. Create Landing Zone Account

Executed only when using AWS Landing Zone baseline

This step starts the `ASEA-ALZCreateAccount_sm` state machine. This state machine is responsible for creating an account using the AVM and waits for the account to be created and configured.

3.2.7. Organizational Unit (OU) Validation

Executed only when using AWS Organizations baseline

This step validates that the OU structure defined in the Accelerator configuration file matches the OU and account structure defined in AWS Organizations. Missing OUs are created. If any AWS Accounts or OUs have been renamed, this step updates the configuration file with the latest information. Accounts located in the incorrect top-level OU cause a state machine failure.

3.2.8. Load Organization Configuration

Executed only when using AWS Organizations baseline

This step calls a Lambda function that validates the Accelerator deployment and configuration.

This step fails when

- an organizational unit that is used by the Accelerator can not be found;
- an account is present in the Accelerator configuration but does not exist in AWS Organizations;
- an account has a name, email address or organizational unit that is different from the Accelerator configuration.
- **returns accounts that have not been created**
- duplicates some validation functionality

3.2.9. Install CloudFormation Role in root

Executed only when using AWS Organizations baseline

This step starts the `ASEA-InstallCfnRoleMaster_sm` state machine. This state machine is responsible for creating the IAM role defined in `organization-admin-role` (default: `AWSCloudFormationStackSetAdministrationRole`) in the root account. You can read more about why this role is created here.

3.2.10. Create Organization Account

Executed only when using AWS Organizations baseline

This step starts the `ASEA-OrgCreateAccount_sm` state machine. This state machine is responsible for creating an account using the AWS Organizations and waits for the account to be created and configured. The newly created account will be moved to the correct organizational unit and will have an the `ASEA-Quarantine-New-Object` Service Control Policy (SCP) attached to it. The SCP will only be removed after the Accelerator state machine has completed.

Application of this SCP ensures that new AWS accounts cannot be used until they have been properly configured and had all Accelerator guardrails applied per the configuration file. It is important that guardrails be consistently in place from the time a user first accesses a new AWS account, until the account is decommissioned. Should the state machine fail, still be in-progress, or has not yet processed the account (for whatever reason), users who access it will be prevented from doing *anything*.

3.2.11. Load Organizational Units

This step calls a Lambda function that loads all Organizational Units. The Organizational Units are stored in the DynamoDB table `ASEA-Parameters` in the root account. The Organizational Units will later be used to determine which configuration file options apply to an AWS account.

3.2.12. Load Accounts

This step calls a Lambda function that loads all accounts. The accounts are stored in the DynamoDB table `ASEA-Parameters` in the root account. The accounts will later be used to deploy stacks in the correct accounts.

3.2.13. Install Execution Roles

This step calls a Lambda function that creates stack sets in all Accelerator accounts. This stack sets contains a single resource, i.e. an IAM role `ASEA-PipelineRole` that can be assumed by the `ASEA-L-SFN-MasterRole`. This IAM role allows the root account to administer the Accelerator accounts.

3.2.14. Delete Default VPCs

This step starts the `ASEA-DeleteDefaultVpcs_sfn` state machine. This state machine is responsible for deleting default subnets, internet gateways and VPCs for all regions and accounts in the Accelerator configuration.

This step fails when one or more default VPCs cannot be deleted. This step does not fail on the first error, it executes on all accounts/regions and then fails with a complete list of errors.

3.2.15. Load Limits

This step calls a Lambda function that loads service quotas and requests a service quota increase according to the configuration file. When a service quota increase request has been closed but not increased, then the service quota request will be issued again when the creation of the last request was at least two days ago.

3.2.16. Enable Trusted Access for Services

This step calls a Lambda function that is responsible for

- enabling AWS service access in the organization;
- enabling AWS Resource Access Manager sharing in the organization;
- creating a service-linked role for AWS IAM Access Analyzer;
- setting the security account as delegated administrator for AWS Firewall Manager;
- setting the security account as delegated administrator for AWS IAM Access Analyzer;
- setting the security account as delegated administrator for Amazon GuardDuty.

3.2.17. Store All Phase Outputs

This step only executes on the first run of the state machine after it has been upgraded to v1.2.0 or above. This step exists solely to support upgrades from Accelerator versions prior to v1.2.0 and can be removed when no existing customers are running versions older than v1.2.0. This steps populates the DynamoDB Outputs table with the outputs from previous executions which were previously stored in S3 (and at one time even stored in secrets manager).

3.2.18. Deploy Phase -1 (Negative one)

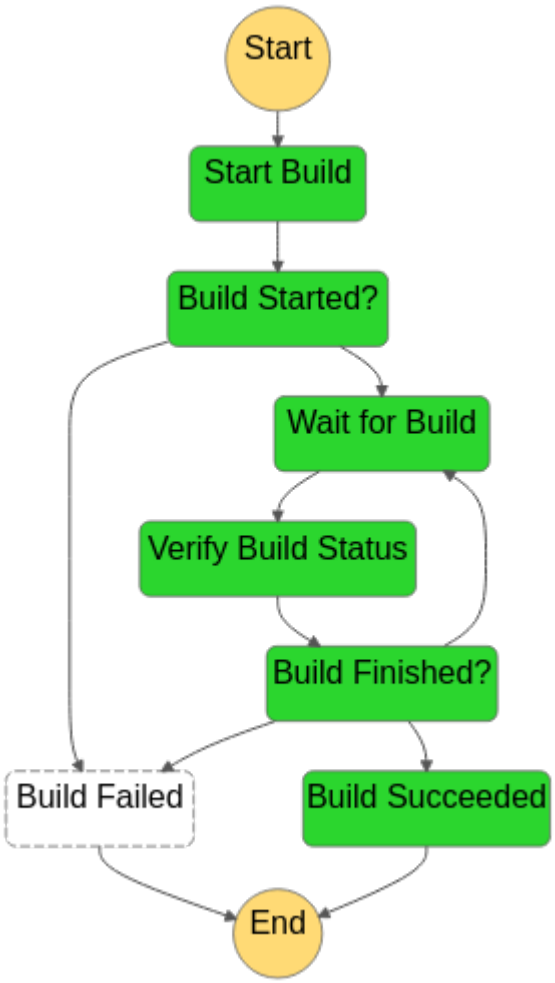
- The following resources are deployed in phase -1:
- Creating required roles for macie custom resources
- Creating required roles for guardDuty custom resources
- Creating required roles for securityHub custom resources
- Creating required roles for IamCreateRole custom resource
- Creating required roles for createSSMDocument custom resource
- Creating required roles for createLogGroup custom resource
- Creating required roles for CWLCentralLoggingSubscriptionFilterRole custom resource
- Creating required roles for TransitGatewayCreatePeeringAttachment custom resource
- Creating required roles for TransitGatewayAcceptPeeringAttachment custom resource
- Creating required roles for createLogsMetricFilter custom resource
- Creating required roles for SnsSubscriberLambda custom resource

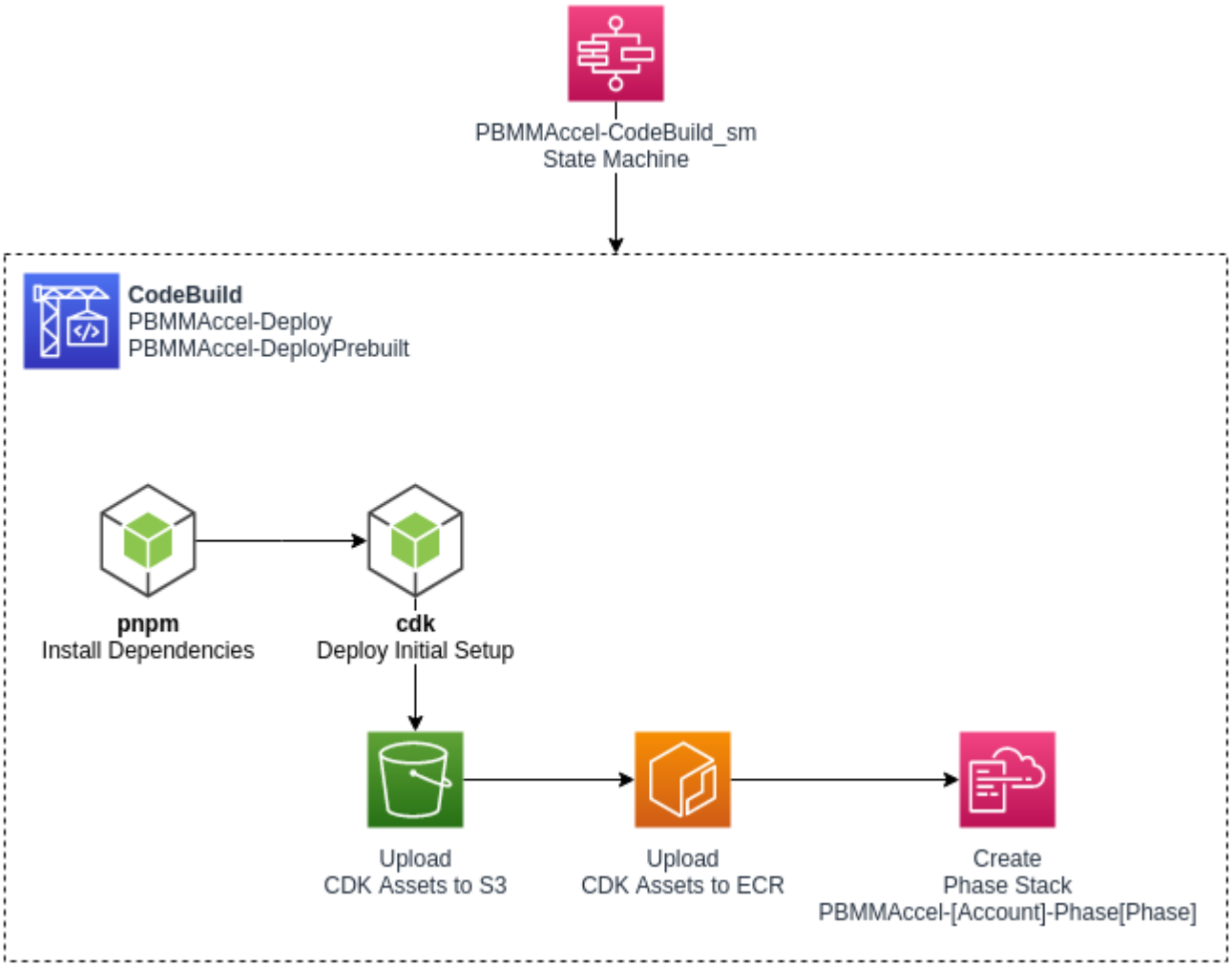
3.2.19. Store Phase -1 Output

This step calls a Lambda function that stores the outputs from the deployed stacks in sub-accounts in the DynamoDB table `ASEA-Outputs` in AWS Secrets Manager in the root account.

3.2.20. Deploy Phase 0

This step starts the `ASEA-CodeBuild_sm` state machine. This state machine is responsible for starting a CodeBuild execution that deploys CloudFormation stacks in Accelerator-managed accounts using CDK.


Start
Start Build
Build Started?
Wait for Build
Verify Build Status
Build Finished?
Build Failed
Build Succeeded
End



The same CodeBuild project is used to deploy all phases. The environment variable `ACCELERATOR_PHASE` in the CodeBuild execution decides which phase to deploy.

The following resources are deployed in phase 0:

- create default EBS encryption key;
- create an AWS log bucket with encryption key;
- create the central log services bucket with encryption key;
- create the Accelerator configuration bucket with encryption key;
- copy artifacts to the Accelerator configuration bucket:
- SCPs;
- firewall configuration;
- account warming (step 1);
- set password policy (step 1);
- create IAM users (step 1):
- create passwords and store in Secrets Manager;
- create MAD deployment (step 1):
- create passwords and store in Secrets Manager;
- create service-linked role;
- create `rsyslog` deployment (step 1);
- create firewalls (step 1);
- create budgets (step 1);
- create transit gateways (step 1);
- create Route53 DNS logging log group;
- enable Macie (step 1);
- enable GuardDuty;
- enable Access Analyzer;

3.2.21. Store Phase 0 Output

This step calls a Lambda function that stores the outputs from the deployed stacks in sub-accounts in the secret `ASEA-Outputs` in AWS Secrets Manager in the root account.

3.2.22. Verify Files

This step verifies that all required files to complete the installation have been provided by the user. If any required files have not been provided, this step will fail and provide a list of *all* missing files.

Why do we wait until so late in the state machine execution to perform this task?

1. we do not want to add or delete files from a customer supplied bucket
2. we do not want to force customers to need to copy and supply standard configuration files for 3. prescriptive installation
3. we want to allow customers to override the sample or standard configuration files
4. the config bucket is only created in Phase 0
5. we copy all the relevant sample files from reference-artifacts folders to our config bucket, and then, we copy any customer supplied config files over top of our sample files
6. This allows customers to override the sample configs, with customer custom configs without needing to either fork the repo, or supply the files on a vanilla install
7. Until we do this file copy, we cannot be sure that the combination of customer supplied and reference-artifact sample files provides all the required files.

3.2.23. Create Config Recorders

This step starts the `ASEA-CreateConfigRecorder_sfn` state machine. This state machine is responsible for creating Config recorders in all accounts and regions.

3.2.24. Add SCPs to Organization

This step calls a Lambda function that creates and attaches the SCPs listed in the Accelerator configuration. The SCP policies are loaded from the Accelerator configuration bucket.

This step fails when

- an SCP policy cannot be found in the Accelerator configuration bucket;
- an SCP could not be attached to an organizational unit or account, e.g. when the maximum number of attached SCPs is exceeded

3.2.25. Deploy Phase 1

- Create S3 Bucket in all accounts and replicate to Log Account Bucket
- Deploy VPC:
- Vpc
- Subnets
- Subnet sharing (RAM)
- Route tables
- Internet gateways
- NAT gateways
- Interface endpoints
- Gateway endpoints
- Transit Gateway Attachments
- IAM Role required for VPC Peering Auto accept
- Firewall images subscription check
- Creates the customer gateways for the EIPs of the firewall
- Create IAM Roles, Users in account based on configuration
- Creates the additional budgets for the account stacks.
- Import Certificates
- Setup SSMSessionManagerDocument
- Create Cost and Usage reports
- Enable Macie in root Account
- GuardDuty setup in Security Account
- Setup CWL Central Logging
- Create Roles required for Flow Logs
- Transit Gateway Peering
- Create LogGroup required for DNS Logging

3.2.26. Store Phase 1 Output

See *Deploy Phase 0*.

3.2.27. Account Default Settings

This step calls a Lambda function that

- enables and sets EBS default encryption for all accounts in the Accelerator configuration;
- enables S3 object level ALZ Cloudtrail logging;
- enables Log Insight events;
- enables KMS encryption using the CMK from the central logging account;
- sets AWS Systems Manager Session Manager default configuration in every Accelerator-managed account in every region with a VPC.

3.2.28. Deploy Phase 2

- Create CloudTrail in root account
- Create VPC Peering Connection
- Create Security Groups for shared VPC in sub accounts
- Setup Security Hub in Security Account
- Setup Cross Account CloudWatch logs sharing by creating roles in sub accounts
- Enable VPC FlowLogs
- Create Active Directory (MAD)
- Create Firewall clusters
- Create Firewall Management instance
- Create Transit Gateway Routes, Association and Propagation
- Enable Macie in Security account and Create Members, Update Config
- GuardDuty - Add existing Org accounts as members and allow new accounts to be members and Publish
- Create SNS Topics in Log Account
- TGW Peering Attachments

3.2.29. Store Phase 2 Output

See *Deploy Phase 0*.

3.2.30. Deploy Phase 3

- create peering connection routes;
- create ALB (step 1);
- create `rsyslog` deployment (step 2);
- create hosted zones, resolver rules and resolver endpoints and Share;
- Enable Security Hub and Invite Sub accounts as members;
- TransitGateway Peering attachment and routes;
- Macie update Session;

3.2.31. Store Phase 3 Output

See *Deploy Phase 0*.

3.2.32. Deploy Phase 4

- SecurityHub Disable Controls
- Creates CloudWatch Metrics on LogGroups

- Associate Shared Resolver Rules to VPC
- Associate Hosted Zones to VPC

3.2.33. Store Phase 4 Output

See *Deploy Phase 0*.

3.2.34. Associate Hosted Zones (Step removed in v1.2.1)

This step calls a Lambda function that associates the private zones, all the interface endpoint zones, and the resolver rules with each VPC that leverages endpoint services. This step was removed in v1.2.1 of the Accelerator codebase.

3.2.35. Add Tags to Shared Resources

This step calls a Lambda function that adds tags to shared resources in the share destination account. For example, when a subnet is shared into another account, this step will add the `Name` tag to the subnet in the shared account.

The supported resources are

- VPCs;
- subnets;
- security groups;
- transit gateway attachments.

3.2.36. Enable Directory Sharing

This step calls a Lambda function that shares Managed Active Directory according to the Accelerator configuration. The directory is shared from the source account to the target account. The directory will be accepted in the target account.

3.2.37. Deploy Phase 5

- create Remote Desktop Gateway;
- create launch configuration;
- create autoscaling group;
- enable central logging to S3 (step 2);
- Create CloudWatch Events for moveAccount, policyChanges and createAccount
- Creates CloudWatch Alarms

3.2.38. Create AD Connector

This step starts the `ASEA-DeleteDefaultVpcs_sfn` state machine. This state machine is responsible for creating AD connectors according to the Accelerator configuration.

This step fails when one or more AD connectors failed to be created.

3.2.39. Store Commit ID

This step calls a Lambda function that stores the commit ID of the configuration file for which the state machine ran.

3.2.40. Detach Quarantine SCP

Executed only when using AWS Organizations baseline

This step calls a Lambda function that stores the commit ID for which the state machine just ran.

5.4 4. Troubleshooting

Issues could occur in different parts of the Accelerator. We'll guide you through troubleshooting these issues in this section.

5.4.1 4.1. Components

4.1.1. State Machine

Viewing the step function `Graph inspector` (depicted above in 2.2), the majority of the main state machine has a large colored box around which is the functionality to catch state machine failures `Main Try Catch block to Notify users`. This large outer box will be blue while the state machine is still executing, it will be green upon a successful state machine execution and will turn orange/yellow on a state machine failure.

What if my State Machine fails? Why? Previous solutions had complex recovery processes, what's involved?

If your main state machine fails, review the error(s), resolve the problem and simply re-run the state machine. We've put a huge focus on ensuring the solution is idempotent and to ensure recovery is a smooth and easy process.

Ensuring the integrity of deployed guardrails is critical in operating and maintaining an environment hosting protected data. Based on customer feedback and security best practices, we purposely fail the state machine if we cannot successfully deploy guardrails.

Additionally, with millions of active customers each supporting different and diverse use cases and with the rapid rate of evolution of the AWS platform, sometimes we will encounter unexpected circumstances and the state machine might fail.

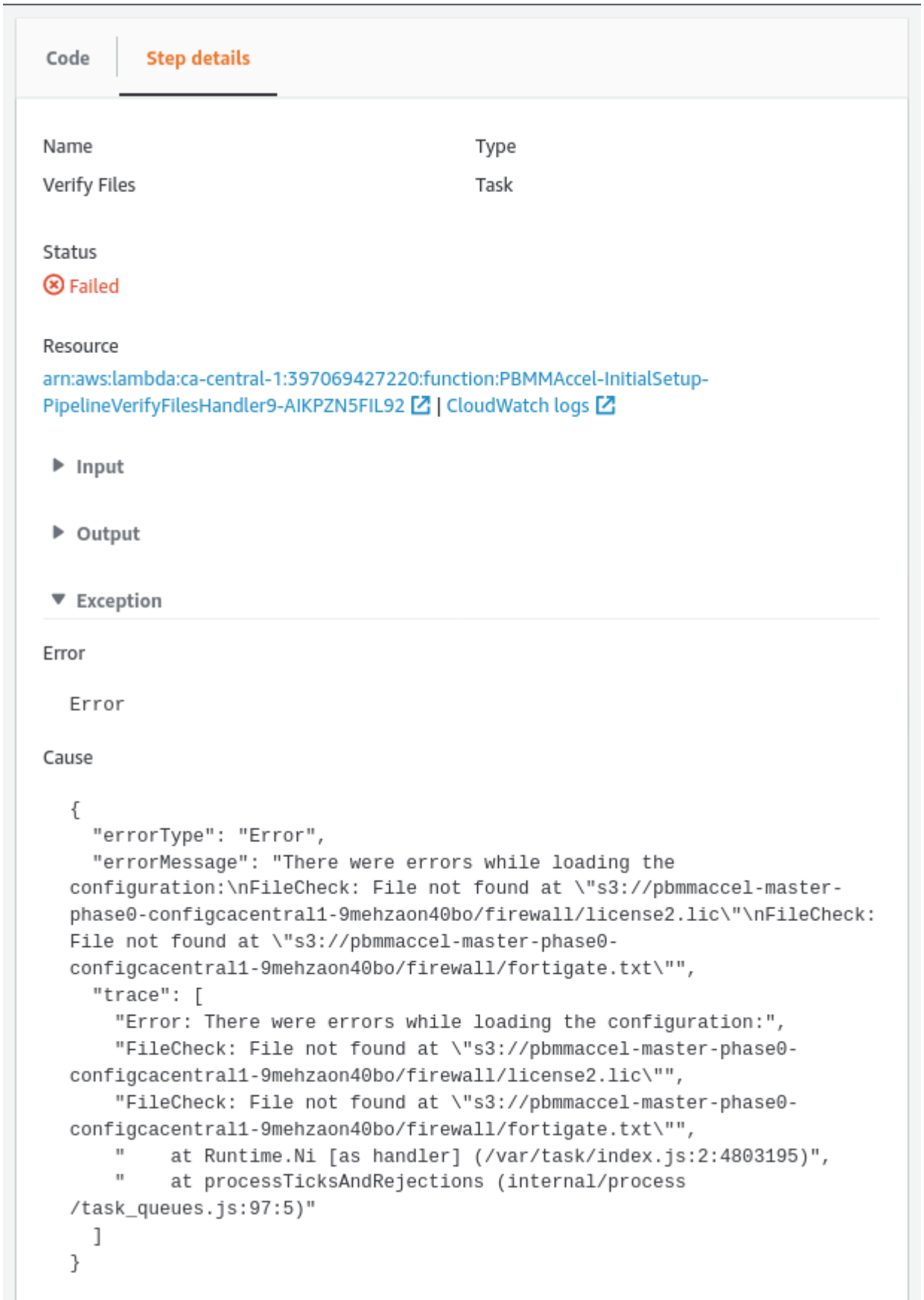
We've spent a lot of time over the course of the Accelerator development process ensuring the solution can roll forward, roll backward, be stopped, restarted, and rerun without issues. A huge focus was placed on dealing with and writing custom code to manage and deal with non-idempotent resources (like S3 buckets, log groups, KMS keys, etc.). We've spent a lot of time ensuring that any failed artifacts are automatically cleaned up and don't cause subsequent executions to fail. We've put a strong focus on ensuring you do not need to go into your various AWS sub-accounts and manually remove or cleanup resources or deployment failures. We've also tried to provide usable error messages that are easy to understand and troubleshoot. As new scenario's are brought to our attention, we continue to adjust the codebase to better handle these situations.

Will your state machine fail at some point in time, likely. Will you be able to easily recover and move forward without extensive time and effort, YES!

As the state machine executes, each step will turn from white (not started), to blue (executing), to green (Success), or grey/red (failure). To diagnose the problem select the grey/red step that failed. If you miss the step and select the outer box, you will have selected the `Main Try Catch block to Notify users`. You need to carefully select the failed step.

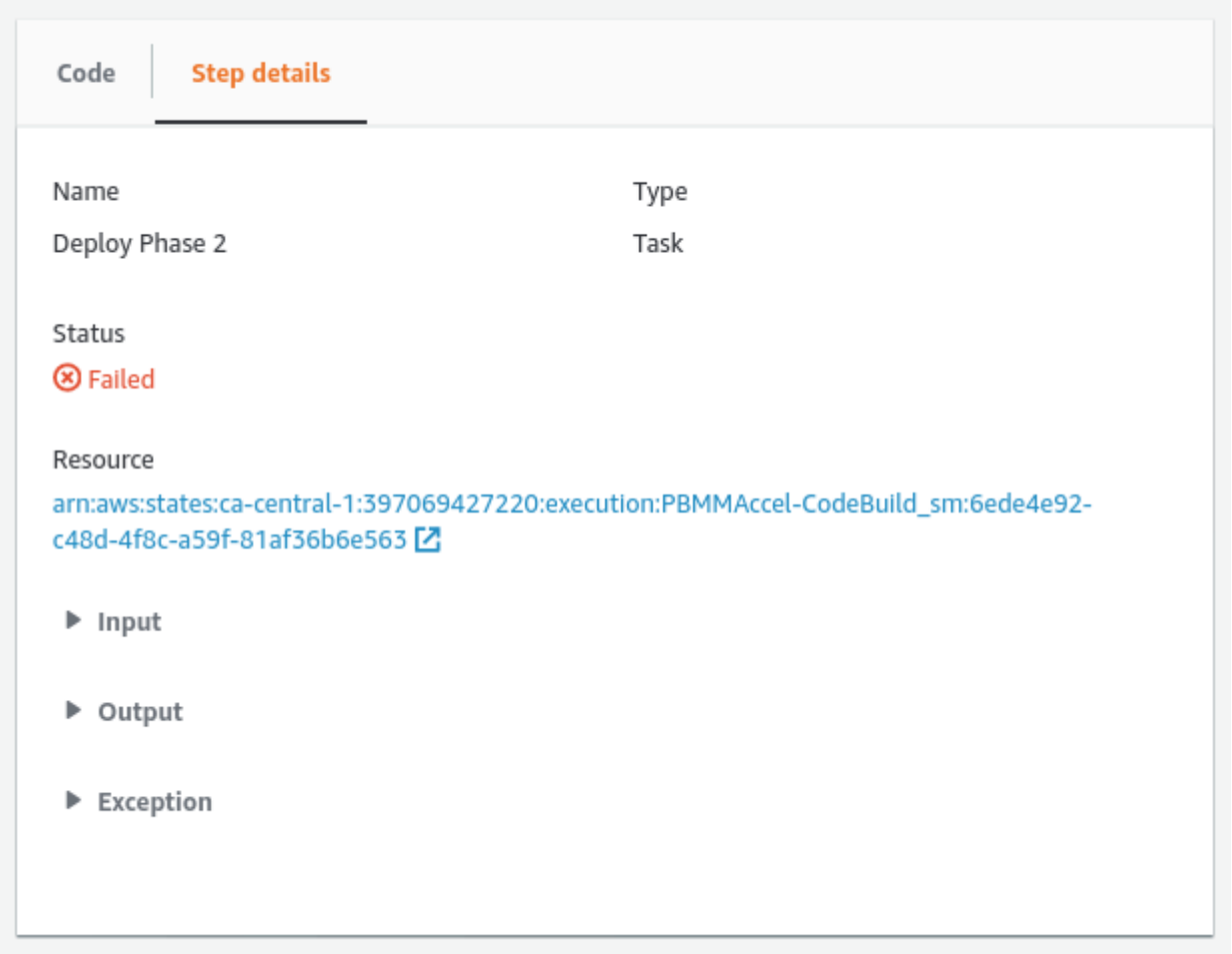
As stated in section 2.2, the state machine contains 3 different types of states, which are each diagnosed differently.

a. If the step is calling a Lambda function then you will see the following after clicking the failed step.



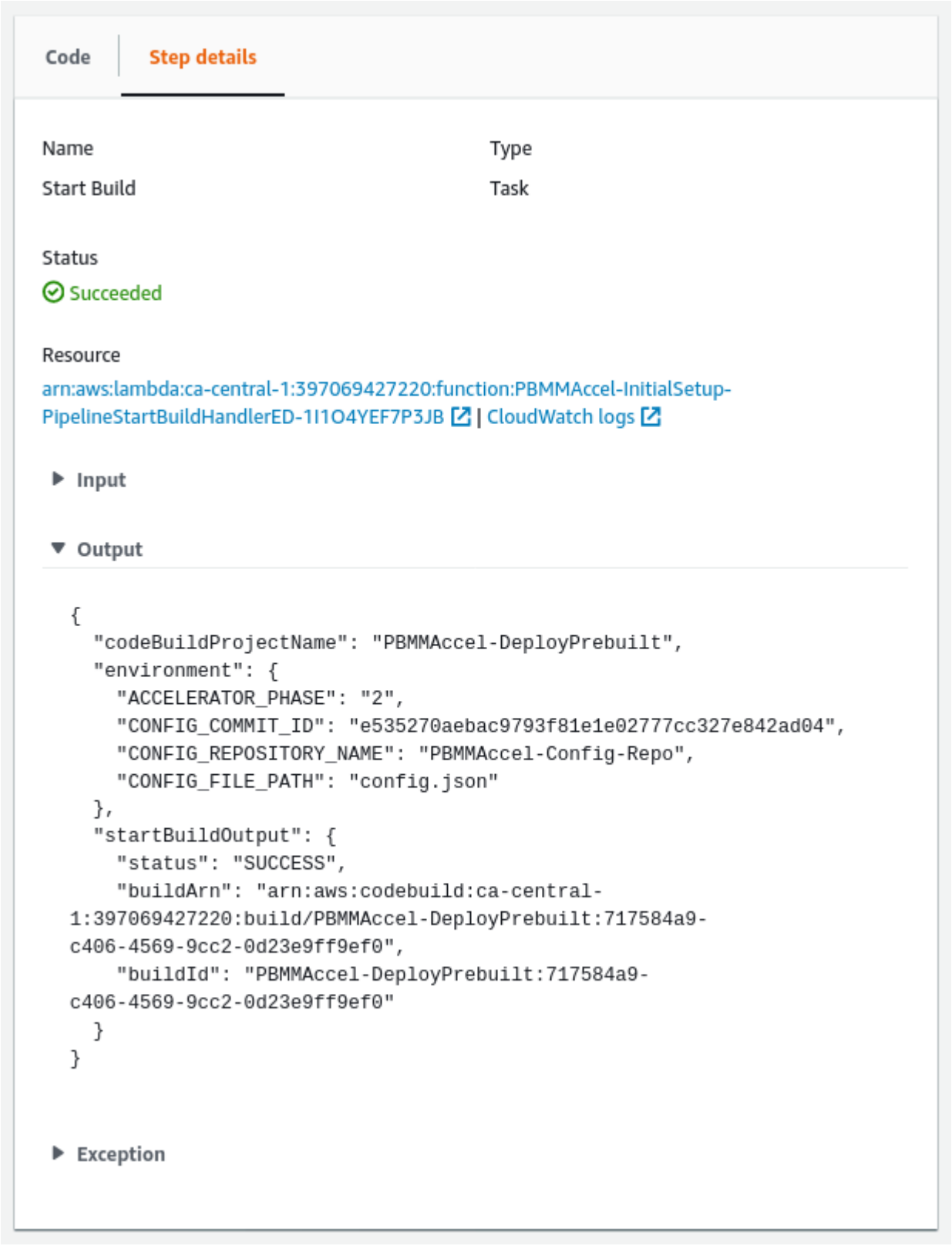
In this case, you can see that the `Cause` section contains a useful message. This message will differ between Lambda functions. In case this message does not make the issue clear, you can click on the `CloudWatch Logs` link in the `Resource` section to view the output of the Lambda function that was called by the step. See the section CloudWatch Logs. Note: The `Resource` section contains two links that blend together. You need to click the second link (`CloudWatch Logs`), not the first link which will open the actual resource/Lambda.

b. In case the failed step started another state machine, you will see the following after clicking the failed step.



To view the state machine execution that failed you can click the link in the `Resource` section.

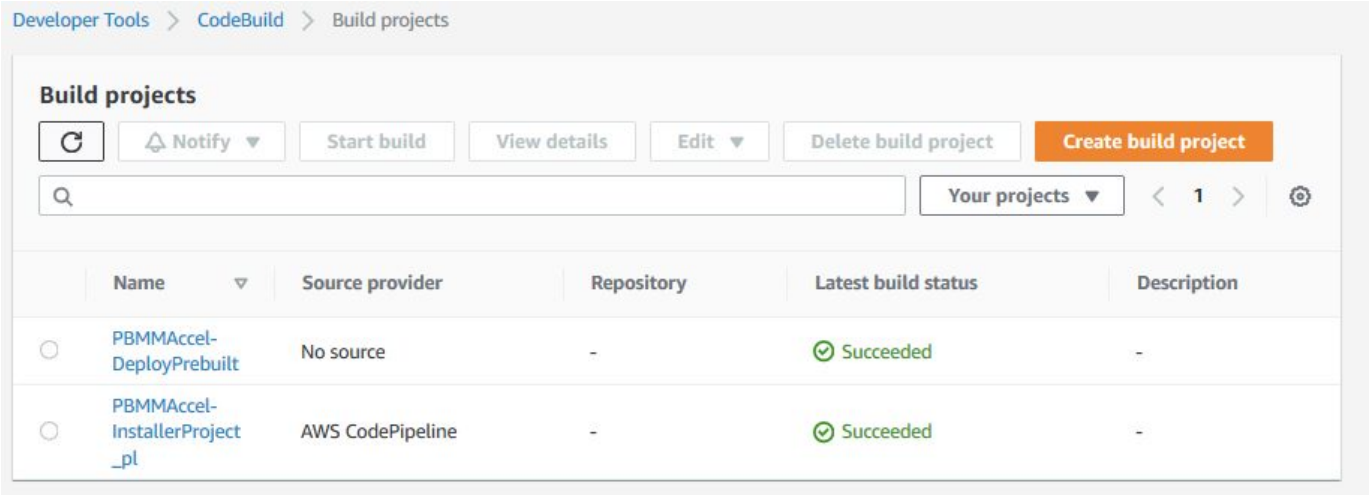
In case the failed step started the CodeBuild state machine, `ASEA-CodeBuild_sm` , you will be able to see the CodeBuild project and execution ID that failed by looking at the output of the `Start Build` step in the `ASEA-CodeBuild_sm` state machine.



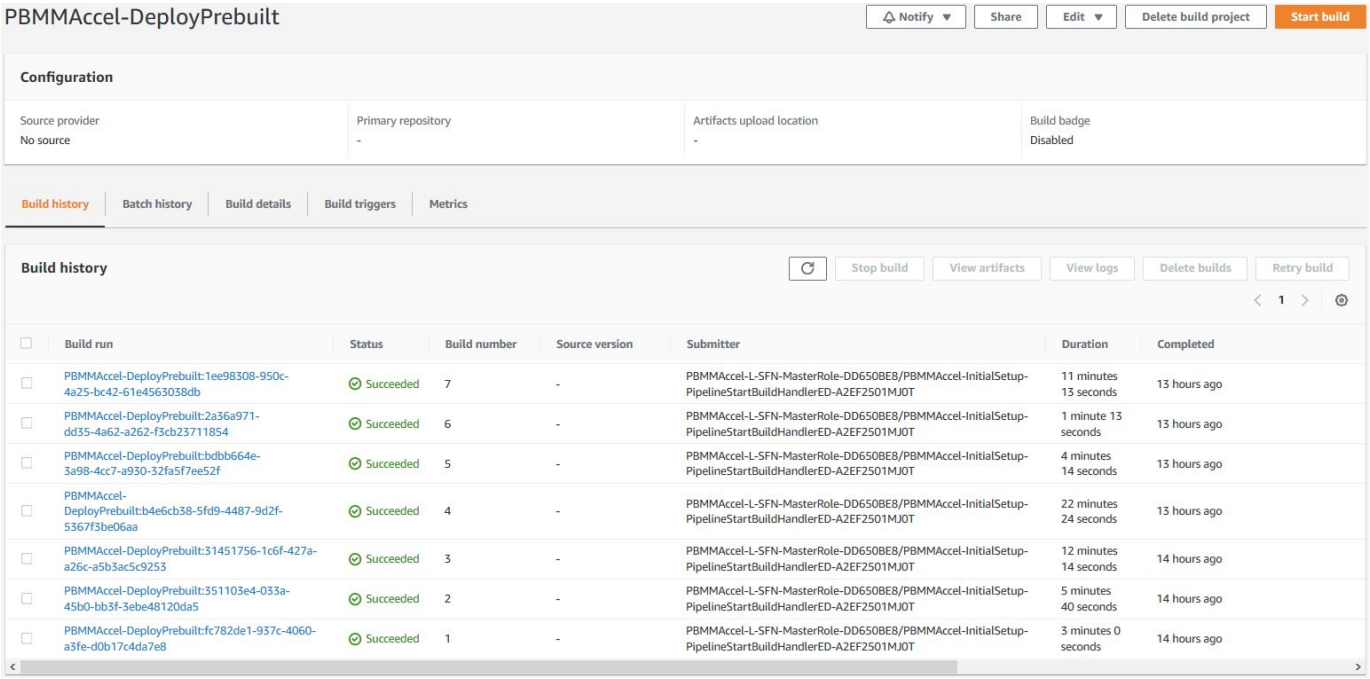
In the image above the execution of CodeBuild project `ASEA-DeployPrebuilt` with ID `ASEA-DeployPrebuilt:717584a9-c406-4569-9cc2-0d23e9ff9ef0` failed. See the CodeBuild section to troubleshoot.

4.1.2. CodeBuild

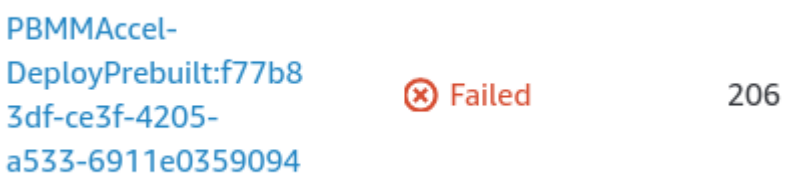
The Accelerator deploys and leverages two CodeBuild projects. The `ASEA-InstallerProject_pl` project is used by the Code Pipeline/Installer stack and `ASEA-DeployPrebuilt` which is used throughout the Accelerator state machine. Both are similar in that they use CDK to deploy stacks. The installer project will not exist, if the installer has been removed.



After a successful installation you will see the following in Codebuild, for the `ASEA-DeployPrebuilt` project:



When an error occurs you will see that the CodeBuild project execution fails when looking in the execution overview.



You can click on the name of the CodeBuild execution and then look inside the logs what caused the failure. These logs can be confusing. We are deploying multiple stacks in parallel and all the messages for all the parallel deployments are interleaved together, so make sure you are correlating the events back to the correct event source. Because we are deploying to 16 regions in parallel, you will also see messages for the same stack deployment interleaved. Even though a task may indicate it is complete and then another seemingly identical task indicates in-progress, the second message is coming from one of the alternate regions.

```
 1/10 | 10:34:30 PM | UPDATE_COMPLETE      | AWS::EC2::Instance               | PerimeterPhase2/FirewallManager (FirewallManagerCCB568C3)
 1/10 | 10:34:30 PM | UPDATE_IN_PROGRESS   | AWS::Lambda::Function            | PerimeterPhase2/Custom::SecurityHubEnableCustomResourceProvider/Handler (CustomSecurityHubEnableCustomResourceProviderHandler2B586BC3)
 1/10 | 10:34:30 PM | UPDATE_COMPLETE      | Custom::S3Template               | PerimeterPhase2/Firewall/Instance1/License/Resource/Default (FirewallInstance1LicenseA9B81640)
 1/10 | 10:34:30 PM | UPDATE_COMPLETE      | Custom::S3Template               | PerimeterPhase2/Firewall/Instance1/Config/Resource/Default (FirewallInstance1ConfigF78EF5CB)
 1/10 | 10:34:30 PM | UPDATE_COMPLETE      | AWS::Lambda::Function            | PerimeterPhase2/Custom::SecurityHubEnableCustomResourceProvider/Handler (CustomSecurityHubEnableCustomResourceProviderHandler2B586BC3)
Stack PBMMAccel-Perimeter-Phase2 is still not stable (UPDATE_ROLLBACK_IN_PROGRESS)
 0/10 | 10:34:32 PM | UPDATE_IN_PROGRESS   | Custom::S3Template               | PerimeterPhase2/Firewall/Instance0/Config/Resource/Default (FirewallInstance0Config984094C3) Requested update required the provider to create a new physical resource
 0/10 | 10:34:32 PM | UPDATE_COMPLETE      | Custom::S3Template               | PerimeterPhase2/Firewall/Instance0/Config/Resource/Default (FirewallInstance0Config984094C3)
 0/10 | 10:34:33 PM | UPDATE_ROLLBACK_COMP | AWS::CloudFormation::Stack       | PBMMAccel-Perimeter-Phase2
 0/10 | 10:34:34 PM | DELETE_IN_PROGRESS   | AWS::CloudFormation::CustomResource | PerimeterPhase2/Firewall/Instance0/Config/Resource/Default (FirewallInstance0Config984094C3)
 0/10 | 10:34:34 PM | DELETE_IN_PROGRESS   | AWS::CloudFormation::CustomResource | PerimeterPhase2/Firewall/Instance1/Config/Resource/Default (FirewallInstance1ConfigF78EF5CB)
 0/10 | 10:34:34 PM | DELETE_IN_PROGRESS   | AWS::CloudFormation::CustomResource | PerimeterPhase2/Firewall/Instance1/License/Resource/Default (FirewallInstance1LicenseA9B81640)
 0/10 | 10:34:34 PM | DELETE_COMPLETE      | AWS::EC2::Instance               | PerimeterPhase2/FirewallManager (FirewallManagerCCB568C3)
 0/10 | 10:34:35 PM | DELETE_IN_PROGRESS   | AWS::CloudFormation::CustomResource | PerimeterPhase2/Firewall/Instance0/License/Resource/Default (FirewallInstance0License05FCD980)
 0/10 | 10:34:35 PM | DELETE_COMPLETE      | AWS::CloudFormation::CustomResource | PerimeterPhase2/Firewall/Instance0/Config/Resource/Default (FirewallInstance0Config984094C3)
Error: Error: The stack named PBMMAccel-Perimeter-Phase2 is in a failed state: UPDATE_ROLLBACK_COMPLETE
    at Object.fulfillAll (/app/initial-setup/templates/promise.ts:8:11)
    at processTicksAndRejections (internal/process/task_queues.js:97:5)
    at CdkToolkit.deployAllStacks (/app/initial-setup/templates/toolkit.ts:143:27)
    at main (/app/initial-setup/templates/cdk.ts:53:21)

[Container] 2020/07/04 22:34:38 Command did not exit successfully sh docker-entrypoint.sh exit status 1
[Container] 2020/07/04 22:34:38 Phase complete: BUILD State: FAILED
[Container] 2020/07/04 22:34:38 Phase context status code: COMMAND_EXECUTION_ERROR Message: Error while executing command: sh docker-entrypoint.sh. Reason: exit status 1
[Container] 2020/07/04 22:34:38 Entering phase POST_BUILD
[Container] 2020/07/04 22:34:38 Phase complete: POST_BUILD State: SUCCEEDED
[Container] 2020/07/04 22:34:38 Phase context status code:  Message:
```

You can for example see the error message `The stack named ASEA-Perimeter-Phase2 is in a failed state: UPDATE_ROLLBACK_COMPLETE`. This means the stack `ASEA-Perimeter-Phase2` failed to update and it had to rollback. The error indicated at the bottom of the Codebuild screen is typically NOT the cause of the failure, just the end result. You need to scroll up and find the FIRST occurrence of an error in the log file. Often starting at the top of the log file and searching for the text `FAIL` (case sensitive), will allow you to find the relevant error message(s) quickly. The failure is typically listed in the CloudFormation update logs.

```
 4/10 | 10:34:22 PM | UPDATE_FAILED        | AWS::EC2::Instance               | PerimeterPhase2/FirewallManager (FirewallManagerCCB568C3) Interface: [eni-0dd94b35aa8be7b3d] in use. (Service: AmazonEC2; Status Code: 400; Error Code: InvalidNetworkInterface.InUse; Request ID: df3728f6-04cb-4fad-97a4-0fd929a27381)
    new FirewallManager (/app/common/constructs/lib/firewall/manager.ts:22:21)
    \_ createFirewallManager (/app/initial-setup/templates/src/deployments/firewall/manager/step-1.ts:93:19)
    \_ Object.step1 (/app/initial-setup/templates/src/deployments/firewall/manager/step-1.ts:55:11)
    \_ deploy (/app/initial-setup/templates/src/apps/phase-2.ts:239:28)
    \_ processTicksAndRejections (internal/process/task_queues.js:97:5)
    \_ Object.deploy (/app/initial-setup/templates/src/app.ts:62:3)
    \_ main (/app/initial-setup/templates/cdk.ts:36:16)
```

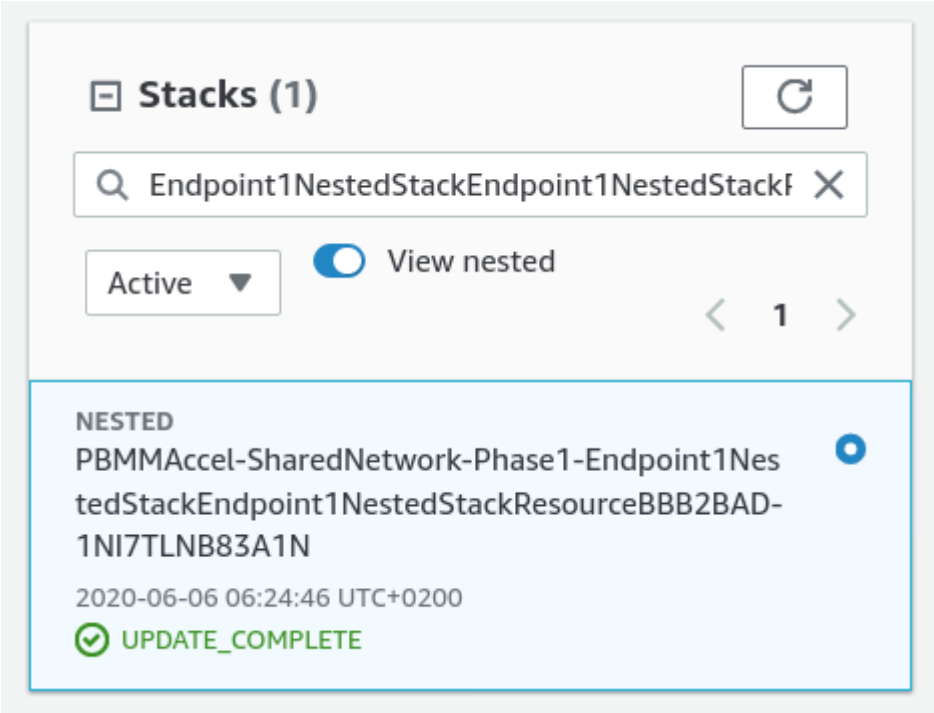
In this example we can see that the resource `FirewallManager` failed to create through CloudFormation. One way to solve this issue is to deprovision the firewall manager in the configuration file and then run the state machine. Next, provision the firewall manager and run the state machine again.

If the error message is not clear, or the error occurred in a nested stack, then a more detailed error will be available in the CloudFormation stack events. See the CloudFormation section below.

```
 1/3 | 10:42:52 PM | CREATE_FAILED        | AWS::CloudFormation::Stack | FunAcctPhase1/VpcStackBrianFunVpc.NestedStack/VpcStackBrianFunVpc.NestedStackResource (VpcStackBrianFunVpcNestedStackVpcStackBrianFunVpcNestedStackResource72499135) Embedded stack arn:aws:cloudformation:ca-central-1:144226684814:stack/PBMMAccel-FunAcct-Phase1-VpcStackBrianFunVpcNestedStackVpcStackBrianFunVpcNestedStackR-1JUUYAR0LBT5J/f40e4700-bb22-11ea-8ced-066237fddb7e was not successfully created: The following resource(s) failed to create: [BrianFunVPCAppFunBrianFunVPCaza339451C7].
    new NestedStack (/app/node_modules/.pnpm/@aws-cdk/core@1.46.0/node_modules/@aws-cdk/core/lib/nested-stack.ts:117:21)
    \_ new NestedStack (/app/node_modules/.pnpm/@aws-cdk/aws-cloudformation@1.46.0/node_modules/@aws-cdk/aws-cloudformation/lib/nested-stack.ts:67:5)
    \_ new VpcStack (/app/initial-setup/templates/src/common/vpc.ts:101:5)
    \_ createVpc (/app/initial-setup/templates/src/apps/phase-1.ts:160:22)
    \_ deploy (/app/initial-setup/templates/src/apps/phase-1.ts:262:17)
    \_ processTicksAndRejections (internal/process/task_queues.js:97:5)
    \_ Object.deploy (/app/initial-setup/templates/src/app.ts:62:3)
    \_ main (/app/initial-setup/templates/cdk.ts:36:16)
```

4.1.3. CloudFormation

In case you want to troubleshoot errors that occurred in CloudFormation, the best way is to look in the CloudFormation stack's events. This requires you to assume a role into the relevant sub-account, and to locate the relevant failed, rolled-back, or deleted stack. Unfortunately, we are unable to log the region of the error message, so depending on what's being deployed, you may need to search all 16 regions for the failed stack.



Events (300+)

Search events

Timestamp	Logical ID	Status	Status reason
2020-06-06 14:35:58 UTC+0200	NotebookepnotebookF1DF839C	DELETE_IN_PROGRESS	-
2020-06-06 14:35:58 UTC+0200	NotebookEndpoint52F5778D	DELETE_COMPLETE	-
2020-06-06 14:35:51 UTC+0200	PBMMAccel-SharedNetwork-Phase1-Endpoint1NestedStackEndpoint1NestedStackResourceBBB2BAD-1NI7TLNB83A1N	UPDATE_ROLLBACK_COMPLETE_CLEANUP_IN_PROGRESS	-
2020-06-06 14:34:38 UTC+0200	PBMMAccel-SharedNetwork-Phase1-Endpoint1NestedStackEndpoint1NestedStackResourceBBB2BAD-1NI7TLNB83A1N	UPDATE_ROLLBACK_IN_PROGRESS	The following resource(s) failed to create: [NotebookEndpoint52F5778D].
2020-06-06 14:34:37 UTC+0200	NotebookEndpoint52F5778D	CREATE_FAILED	Limit of 50 VPC endpoints per VPC exceeded. (Service: AmazonEC2; Status Code: 400; Error Code: VpcEndpointLimitExceeded; Request ID: 76f60902-2ba8-4681-99d0-46fa7d586100)
2020-06-06 14:34:37 UTC+0200	NotebookEndpoint52F5778D	CREATE_IN_PROGRESS	-
2020-06-06 14:34:33 UTC+0200	NotebookepnotebookF1DF839C	CREATE_COMPLETE	-
2020-06-06 14:34:32 UTC+0200	NotebookepnotebookF1DF839C	CREATE_IN_PROGRESS	Resource creation Initiated
2020-06-06 14:34:27 UTC+0200	NotebookepnotebookF1DF839C	CREATE_IN_PROGRESS	-

Load more

When a native resource fails to create or update there are no additional logs available except what is displayed in the `Status reason` column. When a custom resource fails to create or update -- i.e. not a native CloudFormation resource but a resource backed by a custom Lambda function -- then we can find additional logs in CloudWatch.

Often the stack failure occurrs in a managed account instead of the root account. See Switch To a Managed Account to switch to the CloudFormation console in the managed account.

4.1.4. Custom Resource

Custom resources are backed by a Lambda function that implements the creation, modification or deletion or the resource. Every Lambda function has a CloudWatch log group that contains logs about the custom resource creation. To troubleshoot errors in custom resource, you need to check the custom resource's log group.

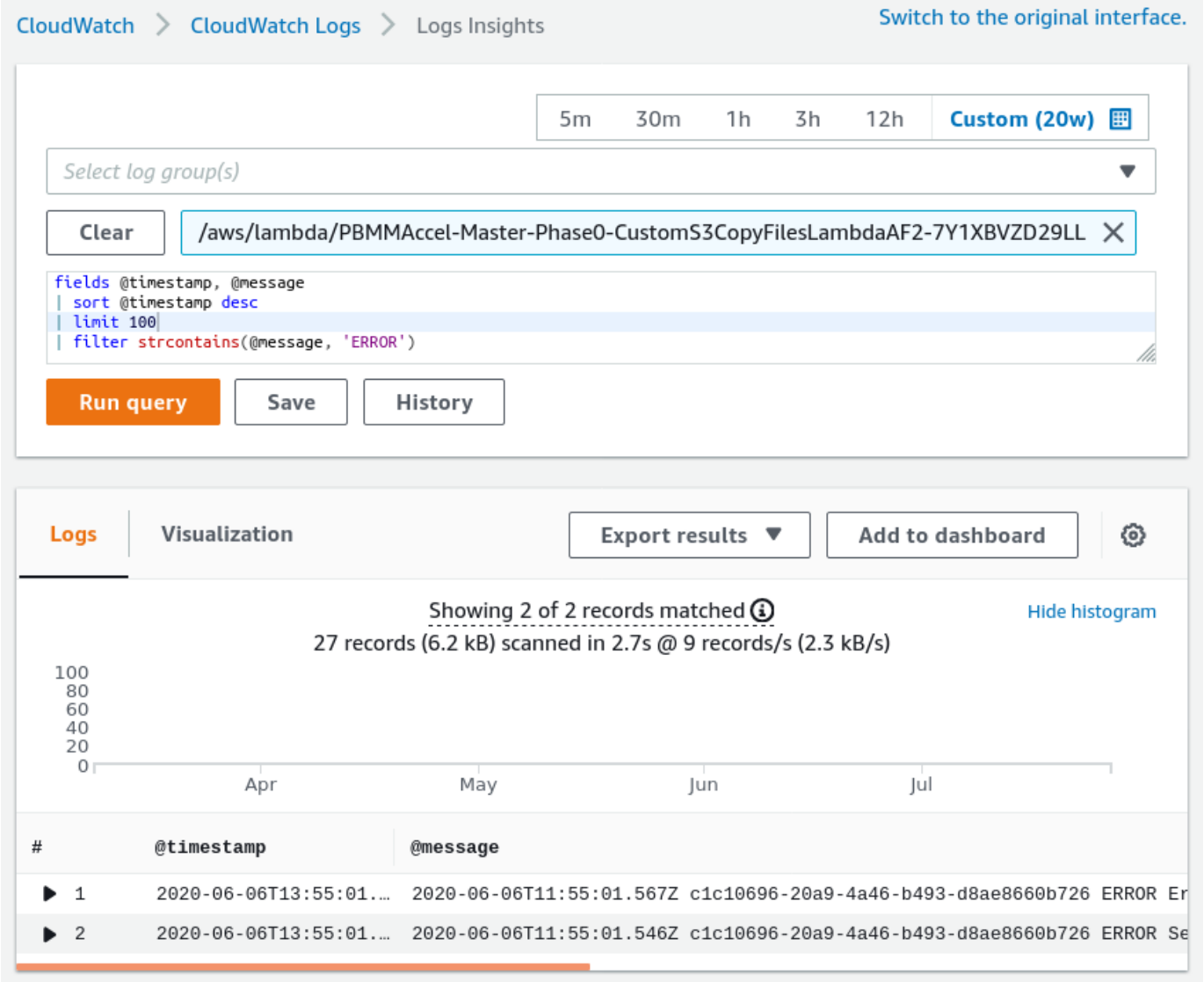
Example custom resource log group names:

```
/aws/lambda/ASEA-Master-Phase1-CustomCurReportDefinitionL-14IHLQCC1LY8L
/aws/lambda/ASEA-Master-Phase2-AWS679f53fac002430cb0da5b7-Z75Q4GG9LIV5
/aws/lambda/ASEA-Operations-Phas-AWS679f53fac002430cb0da5-HMV2YF6OKJET
/aws/lambda/ASEA-Operations-Phas-CustomGetDetectorIdLambd-HEM07DR0DOOJ
```

4.1.5. CloudWatch

When you arrived in CloudWatch logs by clicking on the state machine's step `CloudWatch Logs` link you will immediately see the list of log streams. Every log stream represents an instance of the Lambda function.

You can find errors in multiple log groups using CloudWatch Log Insights.




```
fields @timestamp, @message
| sort @timestamp desc
| filter strcontains(@message, 'ERROR')
| limit 100
```

4.1.6. CodePipeline

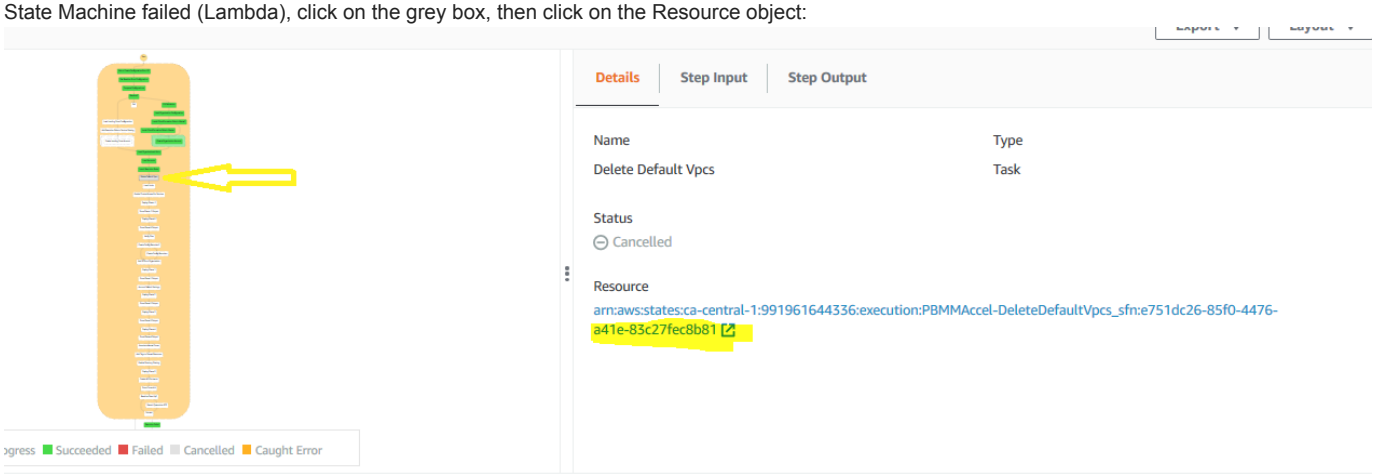
- "Internal Failure" incorrect Github token, repo or branch

5.4.2 4.2. Examples

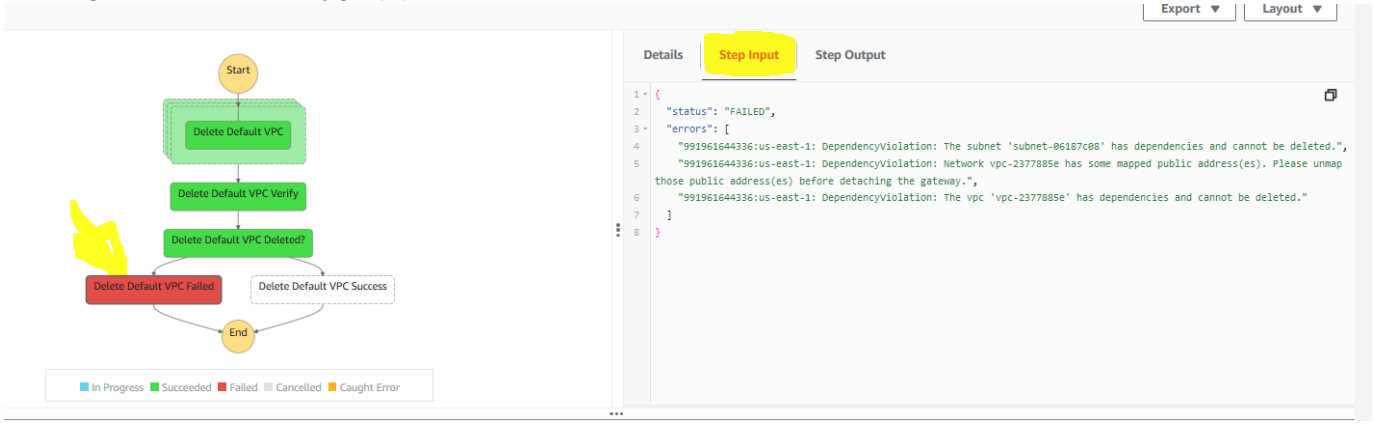
Lets walk through a couple of example:

4.2.1. Example 1

State Machine failed (Lambda), click on the grey box, then click on the Resource object:



Click on the red failed box, click on `Step Input`. The error is clearly indicated, we could not delete a Default VPC because the default VPC had dependencies, in a specified account and region. In this case several dependencies exist and need to be cleaned up to proceed (EIP's and something less obvious like security groups).

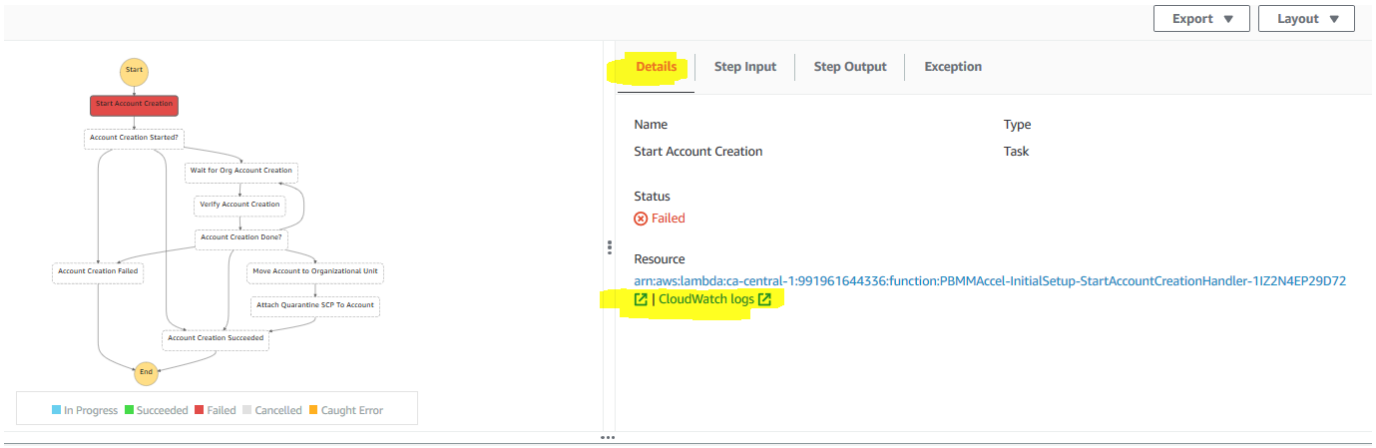


4.2.2. Example 2:

In the next example the state machine failed (sub-state machine) on the create accounts step. In this case rather than clicking on the `Graph inspector` we are going to scroll down through the `Execution event history` underneath the Graph inspector. We are going to find the FIRST failed task from the top of the list and then select the state machine from the prior task:

We will then click onn the red failed box, select `Exception` and we can see a clear error message - we have exceeded the maximum number of AWS accounts allowed in your organization:

Alternatively, in case the `Exception` error is not clear, we can select `Details` and then select `CloudWatch logs` for the **end** of the `Resource` section:

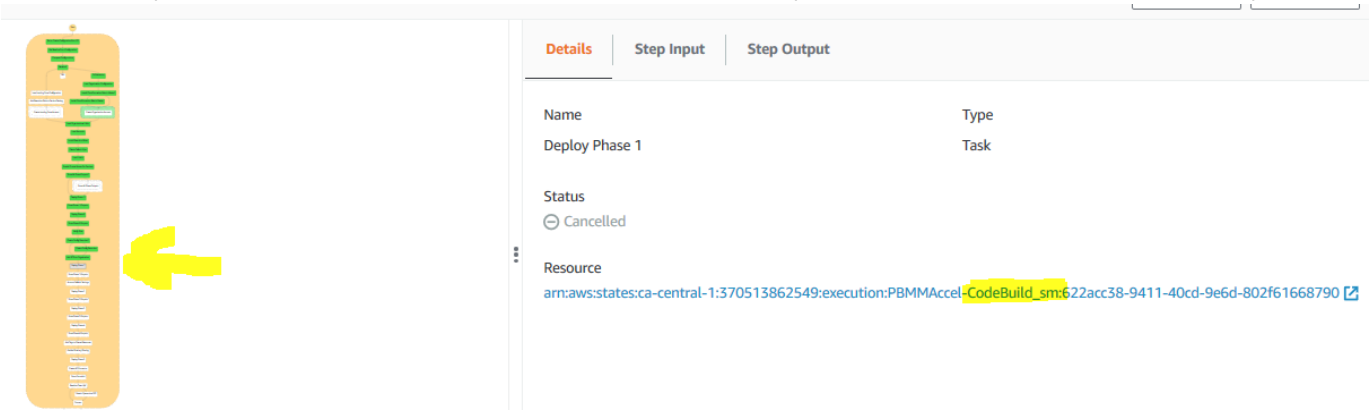


If you open the latest log stream in the opened log group (`/aws/lambda/ASEA-InitialSetup-StartAccountCreationHandler-1IZ2N4EP29D72`) and review the last several messages in the stream, the following clear message also appears:

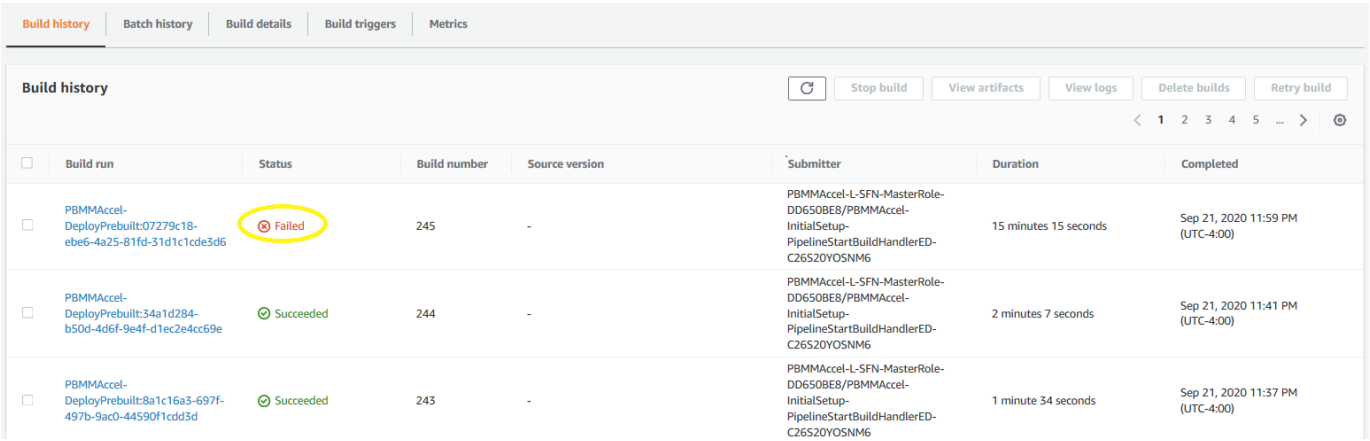
```
2020-09-04T13:39:11.411Z        a66ca383-77a1-4cdf-a431-732e10727cbd    ERROR   Invoke Error
{
    "errorType": "ConstraintViolationException",
    "errorMessage": "You have exceeded the allowed number of AWS accounts.",
    "code": "ConstraintViolationException",
    "message": "You have exceeded the allowed number of AWS accounts.",
    "time": "2020-09-04T13:39:11.410Z",
    "requestId": "2e6e879c-0968-4227-8b7d-61fbdf8abc84",
    "statusCode": 400,
    "retryable": false,
    "retryDelay": 22.311573790276995,
    "stack": [
        "ConstraintViolationException: You have exceeded the allowed number of AWS accounts.",
        "    at constructor.extractError (/var/task/index.js:125:6861)",
        "    at constructor.callListeners (/var/task/index.js:261:31984)",
        "    at constructor.emit (/var/task/index.js:261:31694)",
```

4.2.3. Example 3:

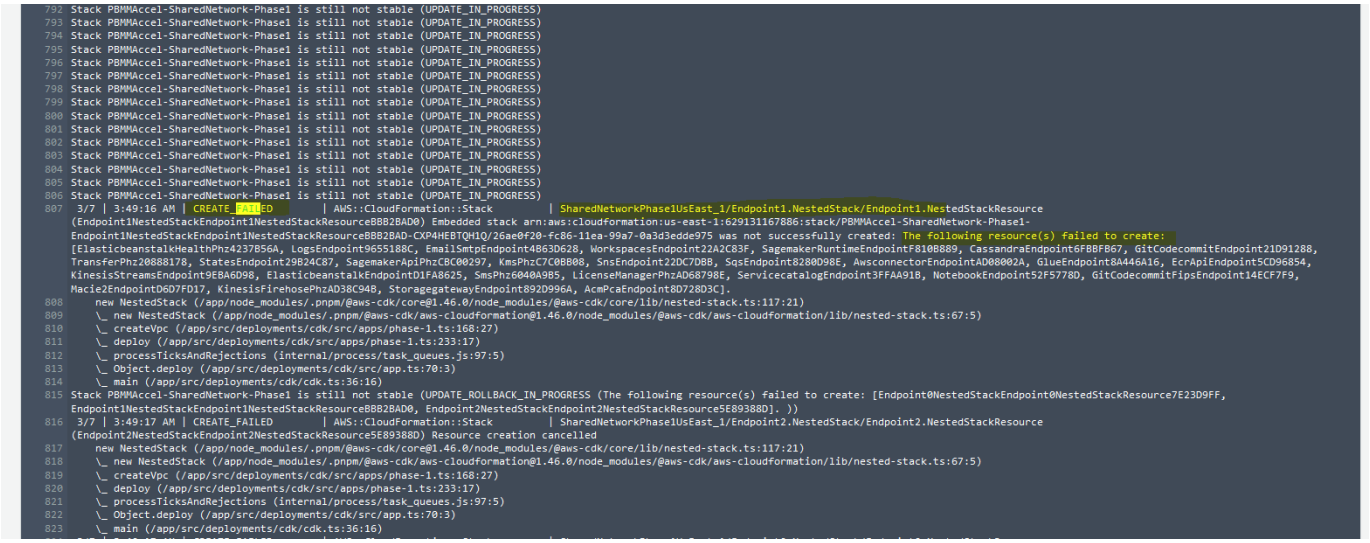
In the next example the state machine failed in one of the CodeBuild state machine steps, based on the `Resource` name of the failed step.



Rather than tracing this failure through the sub-state machine and then into the failed CodeBuild task, we are simply going to open AWS CodeBuild, and open the `ASEA-DeployPrebuilt` task. The failed task should be on the top of the Codebuild `build run` list. Open the build job.



Using your browser, from the top of the page, search for "FAIL", and we are immediately brought to the error. In this particular case we had an issue with the creation of VPC endpoints. We defined something not supported by the current configuration file. The solution was to simply remove the offending endpoints from the config file and re-run the state machine.



5.5 5. How-to

5.5.1 5.1. Restart the State Machine

The state machine can be stopped and restarted at any time. The Accelerator has been design to be able to rollback to a stable state, such that should the state machine be stopped or fail for any reason, subsequent state machine executions can simply proceed through the failed step without manual cleanup or issues (assuming the failure scenario has been resolved). An extensive amount of effort was placed on ensuring seamless customer recovery in failure situations. The Accelerator is idempotent - it can be run as many or as few times as desired with no negative effect. On each state machine execution, the state machine, primarily leveraging the capabilities of CDK, will evaluate the delta's between the old previously deployed configuration and the new configuration and update the environment as appropriate.

The state machine will execute:

- automatically after each execution of the Code Pipeline (new installs, code upgrades, or manual pipeline executions)
- automatically when new AWS accounts are moved into any Accelerator controller OU in AWS Organizations
- when someone manual starts it: `Step Functions`, `ASEA-MainStateMachine_sm`, `Start Execution`, `Start Execution` (leave default values in name and json box)

The state machine prevents users from accidentally performing certain major breaking changes, specifically unsupported AWS platform changes, changes that will fail to deploy, or changes that could be catastrophic to users. If someone knows exactly what they are doing and the full implications of these changes, we provide the option to override these checks. Customers should expect that items we have blocked CANNOT be changed after the Accelerator installation.

These flags should be used with extreme caution. Specifying any of these flags without proper guidance will likely leave your Accelerator in a state of disrepair. These flags were added for internal purposes only - we do NOT support customers providing these flags.

Providing this parameter to the state machine overrides *all* checks:

```
{
   "overrideComparison": true
}
```

Providing any one or more of the following flags will only override the specified check(s):

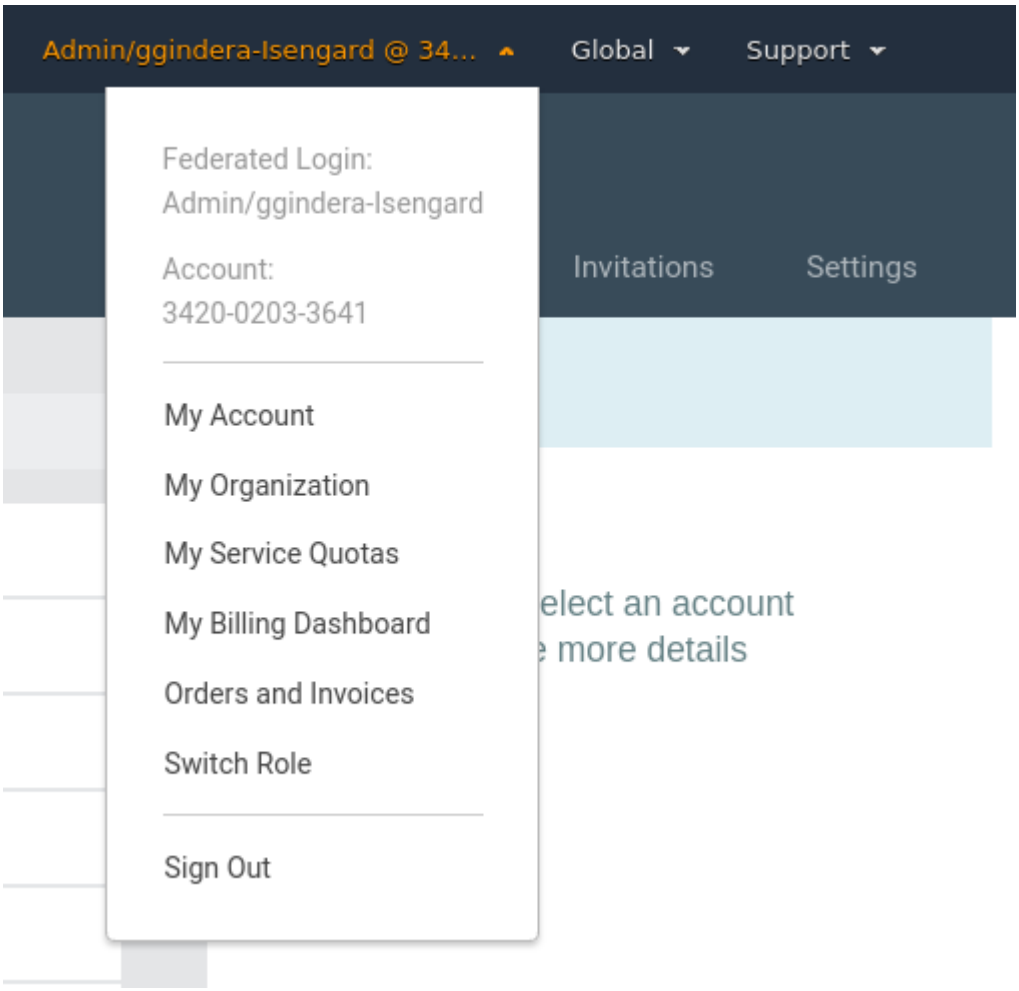
```
{
  "configOverrides": {
    'ov-global-options': true,
    'ov-del-accts': true,
    'ov-ren-accts': true,
    'ov-acct-email': true,
    'ov-acct-ou': true,
    'ov-acct-vpc': true,
    'ov-acct-subnet': true,
    'ov-tgw': true,
    'ov-mad': true,
    'ov-ou-vpc': true,
    'ov-ou-subnet': true,
    'ov-share-to-ou': true,
    'ov-share-to-accounts': true,
    'ov-nacl': true,
    'ov-nfw': true
   }
}
```

Providing this value allows for the forced rebuilding of the DynamoDB Outputs table:

```
{
  "storeAllOutputs": true
}
```

5.5.2 5.2. Switch To a Managed Account

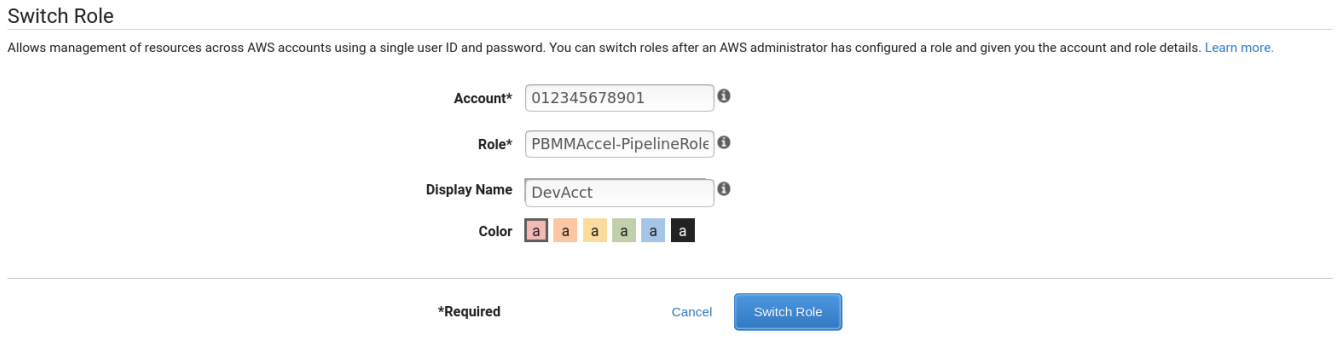
To switch from the root account to a managed account you can click on your account name in the AWS Console. Then choose `Switch Role` in the menu.



In the page that appears next you need to fill out the account ID of the managed account you want to switch to. Next, you need to enter the role name defined in `organization-admin-role` (default: `AWSCloudFormationStackSetAdministrationRole`). And lastly, you need to enter a relevant name so you can later switch roles by using this name.

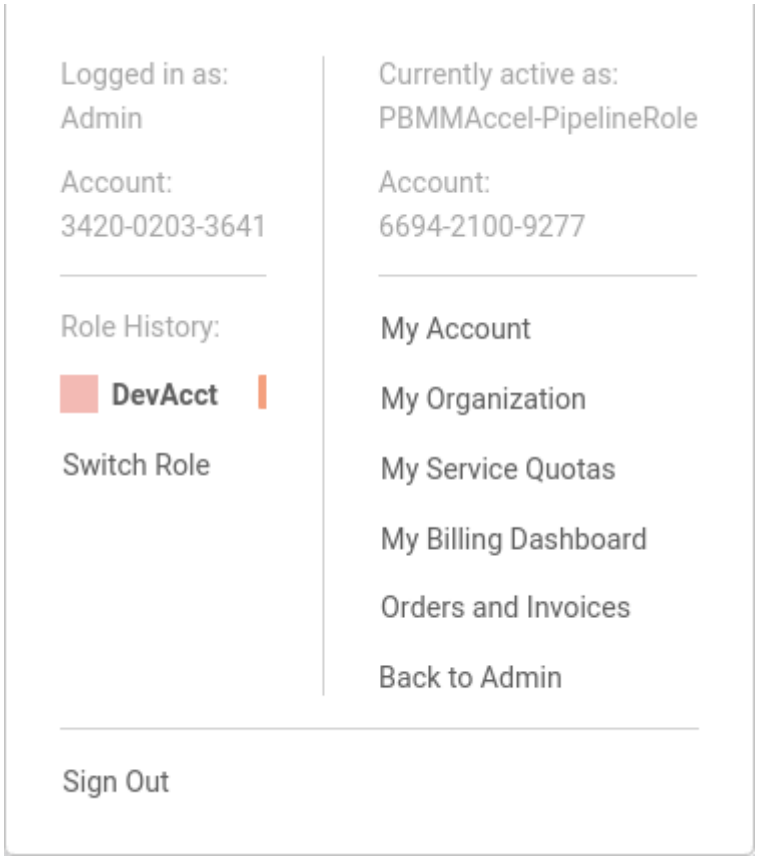
TBD: This role may be locked down starting in v1.2.5 - Update process once direction finalized

Caution: This mechanism is ONLY to be used for troubleshooting Accelerator problems. This role is outside the Accelerator governance process and bypasses **all** the preventative guardrails that protect the Accelerator contructs and prevent users from performing activities in violation of the security guardrails. This role should NOT be used outside this context, all users should be authenticating and logging into the environment through AWS SSO.



After switching to the managed account, the AWS Console header will look like the following image.

You can switch to the same account again quickly by clicking the name you entered previously in the menu.



[1]: https://docs.aws.amazon.com/cdk/latest/guide/home.html

...Return to Accelerator Table of Contents

6. Developer Guide

6.1 Accelerator Developer Guide

This document is a reference document. Instead of reading through it in linear order, you can use it to look up specific issues as needed.

It is important to read the Operations Guide before reading this document. If you're interested in actively contributing to the project, you should also review the Governance and Contributing Guide.

6.2 Development Guide

There are different types of projects in this monorepo.

1. Projects containing CDK code that compiles to CloudFormation templates and deploy to AWS using the CDK toolkit;
2. Projects containing runtime code that is used by the CDK code to deploy Lambda functions;
3. Projects containing reusable code; both for use by the CDK code and/or runtime code.

The CDK code either deploys Accelerator-management resources or Accelerator-managed resources. See the Operations Guide for the distinction between Accelerator-management and Accelerator-managed resources.

The only language used in the project is TypeScript and exceptionally JavaScript. We do not write CloudFormation templates, only CDK code.

When we want to enable functionality in a managed account we try to

1. use native CloudFormation/CDK resource to enable the functionality;
2. create a custom resource to enable the functionality; or
3. lastly create a new step in the `Initial Setup` state machine to enable the functionality.

6.2.1 Project Structure

The folder structure of the project is as follows:

- `src/installer/cdk` : See Installer Stack;
- `src/core/cdk` : See Initial Setup Stack;
- `src/core/runtime` See Initial Setup Stack and Phase Steps and Phase Stacks;
- `src/deployments/runtime` See Phase Steps and Phase Stacks;
- `src/deployments/cdk` : See Phase Steps and Phase Stacks;
- `src/lib/accelerator-cdk` : See Libraries & Tools;
- `src/lib/cdk-constructs` : See Libraries & Tools;
- `src/lib/cdk-plugin-assume-role` : See CDK Assume Role Plugin.
- `src/lib/common-config` : See Libraries & Tools;
- `src/lib/common-outputs` : See Libraries & Tools;
- `src/lib/common-types` : See Libraries & Tools;
- `src/lib/common` : See Libraries & Tools;
- `src/lib/custom-resources/**/cdk` : See Custom Resources;
- `src/lib/custom-resources/**/runtime` : See Custom Resources;

6.2.2 Installer Stack

Read the Operations Guide first before reading this section. This section is a technical addition to the section in the Operations Guide.

As stated in the Operations Guide, the `Installer` stack is responsible for installing the `Initial Setup` stack. It is an Accelerator-management resource. The main resource in the `Installer` stack is the `ASEA-Installer` CodePipeline. The CodePipeline uses this GitHub repository as source action and runs CDK in a CodeBuild step to deploy the `Initial Setup` stack.

```
new codebuild.PipelineProject(stack, 'InstallerProject', {
  buildSpec: codebuild.BuildSpec.fromObject({
    version: '0.2',
    phases: {
      install: {
        'runtime-versions': {
          nodejs: 14,
        },
        // The flag '--unsafe-perm' is necessary to run pnpm scripts in Docker
        commands: ['npm install --global pnpm@6.2.3', 'pnpm install --unsafe-perm --frozen-lockfile'],
      },
      pre_build: {
        // The flag '--unsafe-perm' is necessary to run pnpm scripts in Docker
        commands: ['pnpm recursive run build --unsafe-perm'],
      },
      build: {
        commands: [
          'cd src/core/cdk',
          // Bootstrap the environment for use by CDK
          'pnpx cdk bootstrap --require-approval never',
          // Deploy the Initial Setup stack
          'pnpx cdk deploy --require-approval never',
        ],
      },
    },
  }),
});
```

When the CodePipeline finishes deploying the `Initial Setup` stack, it starts a Lambda function that starts the execution of the `Initial Setup` stack's main state machine.

The `Initial Setup` stack deployment receives environment variables from the CodePipeline's CodeBuild step. The most notable environment variables are:

- `ACCELERATOR_STATE_MACHINE_NAME` : The `Initial Setup` will use this name for the main state machine. So it is the `Installer` stack that decides the name of the main state machine. This way we can confidently start the main state machine of the `Initial Setup` stack from the CodePipeline;
- `ENABLE_PREBUILT_PROJECT` : See Prebuilt Docker Image.

6.2.3 Initial Setup Stack

Read Operations Guide first before reading this section. This section is a technical addition to the section in the Operations Guide.

As stated in the Operations Guide, the `Initial Setup` stack consists of a state machine, named `ASEA-MainStateMachine_sm`, which executes steps to create the Accelerator-managed stacks and resources in the managed accounts. It is an Accelerator-management resource.

The `Initial Setup` stack is defined in the `src/core/cdk` folder.

The `Initial Setup` stack is similar to the `Installer` stack, as in that it runs a CodeBuild project to deploy others stacks using CDK. In case of the `Initial Setup` stack

- we use a AWS Step Functions State Machine to run steps instead of using a CodePipeline;
- we deploy multiple stacks, called `Phase` stacks, in Accelerator-managed accounts. These `Phase` stacks contain Accelerator-managed resources.

In order to install these `Phase` stacks in Accelerator-managed accounts, we need access to those accounts. We create a stack set in the Organization Management (root) account that has instances in all Accelerator-managed accounts. This stack set contains what we call the `PipelineRole`.

The code for the steps in the state machine is in `src/core/runtime`. All the steps are in different files but are compiled into a single file. We used to compile all the steps separately but we would hit a limit in the amount of parameters in the generated CloudFormation template. Each step would have its own CDK asset that would introduce three new parameters. We quickly reached the limit of 60 parameters in a CloudFormation template and decided to compile the steps into a single file and use it across all different Lambda functions.

CodeBuild and Prebuilt Docker Image

The CodeBuild project that deploys the different `Phase` stacks is constructed using the `CdkDeployProject` or `PrebuiltCdkDeployProject` based on the value of the environment variable `ENABLE_PREBUILT_PROJECT`.

The first, `CdkDeployProject` constructs a CodeBuild project that copies this whole Github repository as a ZIP file to S3 using CDK S3 assets. This ZIP file is then used as source for the CodeBuild project. When the CodeBuild project executes, it runs `pnpm recursive install` which in turn will run all `prepare` scripts in all `package.json` files in the project -- as described in section CDK Code Dependency on Lambda Function Code.

After installing the dependencies, the CodeBuild project deploys the `Phase` stacks.

```
cd src/deployments/cdk
sh codebuild-deploy.sh
```

We have more than 50 workspace projects in the monorepo with a `prepare` script, so the `pnpm recursive install` step can take some time. Also, the CodeBuild project will run for each deployed `Phase` stack in each Accelerator-managed account.

This is where the `PrebuiltCdkDeployProject` CodeBuild project comes in. The `PrebuiltCdkDeployProject` contains a Docker image that contains the whole project in the `/app` directory and has all the dependencies already installed.

```
FROM node:12-alpine3.11
# Install the package manager
RUN npm install --global pnpm
RUN mkdir /app
WORKDIR /app
# Copy over the project root to the /app directory
ADD . /app/
# Install the dependencies
RUN pnpm install --unsafe-perm --frozen-lockfile
# Build all Lambda function runtime code
RUN pnpm recursive run build --unsafe-perm
```

When this CodeBuild project executes, it uses the Docker image as base -- the dependencies are already installed -- and runs the same commands as the `CdkDeployProject` to deploy the `Phase` stacks.

Passing Data to Phase Steps and Phase Stacks

Some steps in the state machine write data to Amazon DynamoDB. This data is necessary to deploy the `Phase` stacks later on. At one time this data was written to Secrets Manager and/or S3, these mechanisms were deemed ineffective due to object size limitations or consistency challenges and were all eventually migrated to DynamoDB.

- `Load Accounts` step: This step finds the Accelerator-managed accounts in AWS Organizations and stores the account key -- the key of the account in `mandatory-account-configs` or `workload-account-configs` object in the Accelerator config -- and account ID and other useful information in the `ASEA-Parameters` table, `accounts/#` key and `accounts-items-count` key;
- `Load Organizations` step: More or less the same as the `Load Accounts` step but for organizational units in AWS Organizations and stores the values in the `ASEA-Parameters` table, `organizations` key;
- `Load Limits` step: This step requests limit increases for Accelerator-managed accounts and stores the current limits in the the `ASEA-Parameters` table, `limits` key.
- `Store Phase X Output` : This step loads stack outputs from all existing `Phase` stacks and stores the outputs in the DynamoDB table `ASEA-Outputs` .

Other data is passed through environment variables:

- `ACCELERATOR_NAME` : The name of the Accelerator;
- `ACCELERATOR_PREFIX` : The prefix for all named Accelerator-managed resources;
- `ACCELERATOR_EXECUTION_ROLE_NAME` : The name of the execution role in the Accelerator-managed accounts. This is the `PipelineRole` we created using stack sets.

6.2.4 Phase Steps and Phase Stacks

Read Operations Guide first before reading this section. This section is a technical addition to the *Deploy Phase X* sections in the Operations Guide.

The `Phase` stacks contain the Accelerator-managed resources. The reason the deployment of Accelerator-managed resources is split into different phases is because there cannot be cross account/region references between CloudFormation stacks. See Cross-Account/Region References.

The `Phase` stacks are deployed by a CodeBuild project in the `Initial Setup` stack as stated in the previous paragraphs. The CodeBuild project executes the `codebuild-deploy.sh` script. See `initial-setup.ts` .

The `codebuild-deploy.sh` script executes the `cdk.ts` file.

The `cdk.ts` file is meant as a replacement for the `cdk` CLI command. To deploy a phase stack you would **not** run `pnpx cdk deploy` but `cdk.sh --phase 1` . See CDK API for more information why we use the CDK API instead of using the CDK CLI.

The `cdk.ts` command parses command line arguments and creates all the `cdk.App` for all accounts and regions for the given `--phase` . When you pass the `--region` or `--account-key` command, all the `cdk.App` for all accounts and regions will still be created, except that only the `cdk.App` s matching the parameters will be deployed. This behavior could be optimized in the future. See Stacks with Same Name in Different Regions for more information why we're creating multiple `cdk.App` s.

6.2.5 Store outputs to SSM Parameter Store

Customers need the CloudFormation outputs of resources that are created by the accelerator in order to deploy their own resources in AWS. eg. vpcId in shared-network account to create an ec2 instance, etc.

This step loads the stack outputs from our DynamoDB Table `ASEA-Outputs` and stores as key value pairs in SSM Parameter Store in each account.

Example values are

- /ASEA/network/vpc/1/name => Endpoint
- /ASEA/network/vpc/1/id => vpc-XXXXXXXXXX

`ASEA-Outputs-Utils` DynamoDB Table is used extensively to maintain same index irrespective of configuration changes.

This allows customers to reliably build Infrastructure as Code (IaC) which depends on accelerator deployed objects like VPC's, security groups, subnets, ELB's, KMS keys, IAM users and policies. Rather than making the parameters dependent on object names, we used an indexing scheme, which we maintain and don't update as a customers configuration changes. We have attempted to keep the index values consistent across accounts (based on the config file), such that when code is propoted through the SDLC cycle from Dev to Test to Prod, the input parameters to the IaC scripts do not need to be updated, the App subnet, for example, will have the same index value in all accounts.

Phases and Deployments

The `cdk.ts` file calls the `deploy` method in the `apps/app.ts` . This `deploy` method loads the Accelerator configuration, accounts, organizations from DynamoDB; loads the stack outputs from Amazon DynamoDB; and loads required environment variables.

```
/**
 * Input to the `deploy` method of a phase.
 */
export interface PhaseInput {
  // The config.json file
  acceleratorConfig: AcceleratorConfig;
  // Auxiliary class to construct stacks
  accountStacks: AccountStacks;
  // The list of accounts, their key in the configuration file and their ID
  accounts: Account[];
  // The parsed environment variables
  context: Context;
  // The list of stack outputs from previous phases
  outputs: StackOutput[];
  // Auxiliary class to manage limits
  limiter: Limiter;
}
```

It is important to note that no configuration is hard-coded. The CloudFormation templates are generated by CDK and the CDK constructs are created according to the configuration file. Changes to the configuration will change the CDK construct tree and that will result in a different CloudFormation template that is deployed.

The different phases are defined in `apps/phase-x.ts` . Historically we created all CDK constructs in the `phase-x.ts` files. After a while the `phase-x.ts` files started to get too big and we moved to separating the logic into separate deployments. Every logical component has a separate folder in the `deployments` folder. Every `deployment` consists of so-called steps. Separate steps are put in loaded in phases.

For example, take the `deployments/defaults` deployment. The deployment consists of two steps, i.e. `step-1.ts` and `step-2.ts`. `deployments/defaults/step-1.ts` is created in `apps/phase-0.ts` and `deployments/defaults/step-2.ts` is created in `apps/phase-1.ts`. You can find more details about what happens in each phase in the Operations Guide.

`apps/phase-0.ts`

```
export async function deploy({ acceleratorConfig, accountStacks, accounts, context }: PhaseInput) {
  // Create defaults, e.g. S3 buckets, EBS encryption keys
  const defaultsResult = await defaults.step1({
    acceleratorPrefix: context.acceleratorPrefix,
    accountStacks,
    accounts,
    config: acceleratorConfig,
  });
```

`apps/phase-1.ts`

```
export async function deploy({ acceleratorConfig, accountStacks, accounts, outputs }: PhaseInput) {
  // Find the central bucket in the outputs
  const centralBucket = CentralBucketOutput.getBucket({
    accountStacks,
    config: acceleratorConfig,
    outputs,
  });

  // Find the log bucket in the outputs
  const logBucket = LogBucketOutput.getBucket({
    accountStacks,
    config: acceleratorConfig,
    outputs,
  });

  // Find the account buckets in the outputs
  const accountBuckets = await defaults.step2({
    accounts,
    accountStacks,
    centralLogBucket: logBucket,
    config: acceleratorConfig,
  });
}
```

Passing Outputs between Phases

The CodeBuild step that is responsible for deploying a `Phase` stack runs in the Organization Management (root) account. We wrote a CDK plugin that allows the CDK deploy step to assume a role in the Accelerator-managed account and create the CloudFormation `Phase` stack in the managed account. See CDK Assume Role Plugin.

After a `Phase-X` is deployed in all Accelerator-managed accounts, a step in the `Initial Setup` state machine collects all the `Phase-X` stack outputs in all Accelerator-managed accounts and regions and stores theses outputs in DynamoDB.

Then the next `Phase-X+1` deploys using the outputs from the previous `Phase-X` stacks.

See Creating Stack Outputs for helper constructs to create outputs.

Decoupling Configuration from Constructs

At the start of the project we created constructs that had tight coupling to the Accelerator config structure. The properties to instantiate a construct would sometimes have a reference to an Accelerator-specific interface. An example of this is the `Vpc` construct in `src/deployments/cdk/common/vpc.ts`.

Later on in the project we started decoupling the Accelerator config from the construct properties. Good examples are in `src/lib/cdk-constructs/`.

Decoupling the configuration from the constructs improves reusability and robustness of the codebase.

6.2.6 Libraries & Tools

CDK Assume Role Plugin

At the time of writing, CDK does not support cross-account deployments of stacks. It is possible however to write a CDK plugin and implement your own credential loader for cross-account deployment.

We wrote a CDK plugin that can assume a role into another account. In our case, the Organization Management (root) account will assume the `PipelineRole` in an Accelerator-managed account to deploy stacks.

CDK API

We use the internal CDK API to deploy the `Phase` stacks instead of the CDK CLI for the following reasons:

- It allows us to deploy multiple stacks in parallel;
- Disable stack termination before destroying a stack;
- Delete a stack after it initially failed to create;
- Deploy multiple apps at the same time -- see Stacks with Same Name in Different Regions.

The helper class `CdkToolkit` in `toolkit.ts` wraps around the CDK API.

The risk of using the CDK API directly is that the CDK API can change at any time. There is no stable API yet. When upgrading the CDK version, the `CdkToolkit` wrapper might need to be adapted.

AWS SDK Wrappers

You can find `aws-sdk` wrappers in the `src/lib/common/src/aws` folder. Most of the classes and functions just wrap around `aws-sdk` classes and implement promises and exponential backoff to retryable errors. Other classes, like `Organizations` have additional functionality such as listing all the organizational units in an organization in the function `listOrganizationalUnits`.

Please use the `aws-sdk` wrappers throughout the project or write an additional wrapper when necessary.

Configuration File Parsing

The configuration file is defined and validated using the `io-ts` library. See `src/lib/common-config/src/index.ts`. In case any changes need to be made to the configuration file parsing, this is the place to be.

We wrap a class around the `AcceleratorConfig` type that contains additional helper functions. You can add your own additional helper functions.

ACCELERATORNAMETAGGER

`AcceleratorNameTagger` is a CDK aspect that sets the name tag on specific resources based on the construct ID of the resource.

The following example illustrates its purpose.

```
const stack = new cdk.Stack();
new ec2.CfnVpc(stack, 'SharedNetwork', {});
Aspects.of(stack).add(new AcceleratorNameTagger());
```

The example above synthesizes to the following CloudFormation template.

```
Resources:
  SharedNetworkAB7JKF7:
    Properties:
      Tags:
        - Key: Name
          Value: SharedNetwork_vpc
```

ACCELERATORSTACK

`AcceleratorStack` is a class that extends `cdk.Stack` and adds the `Accelerator` tag to all resources in the stack. It also applies the aspect `AcceleratorNameTagger`.

It is also used by the `accelerator-name-generator.ts` functions to find the name of the `Accelerator`.

NAME GENERATOR

The `accelerator-name-generator.ts` file contains methods that create names for resources that are optionally prefixed with the Accelerator name, and optionally suffixed with a hash based on the path of the resource, the account ID and region of the stack.

The functions should be used to create pseudo-random names for IAM roles, KMS keys, key pairs and log groups.

ACCOUNTSTACKS

`AccountStacks` is a class that manages the creation of an `AcceleratorStack` based on a given account key and region. If an account with the given account key cannot be found in the accounts object -- which is loaded by `apps/app.ts` then no stack will be created. This class is used extensively throughout the phases and deployment steps.

```
export async function step1(props: CertificatesStep1Props) {
  const { accountStacks, centralBucket: centralBucket, config } = props;

  for (const { accountKey, certificates } of config.getCertificateConfigs()) {
    if (certificates.length === 0) {
      continue;
    }

    const accountStack = accountStacks.tryGetOrCreateAccountStack(accountKey);
    if (!accountStack) {
      console.warn(`Cannot find account stack ${accountKey}`);
      continue;
    }

    for (const certificate of certificates) {
      createCertificate({
        centralBucket,
        certificate,
        scope: accountStack,
      });
    }
  }
}
```

VPC AND IMPORTEDVPC

`Vpc` is an interface in the `src/lib/cdk-constructs/src/vpc/vpc.ts` file that attempts to define an interface for a VPC. The goal of the interface is to be implemented by a `cdk.Construct` that implements the interface. This CDK issue provides more background [https://github.com/aws/aws-cdk/issues/5927].

Another goal of the interface is to provide an interface on top of imported VPC outputs. This is what the `ImportedVpc` class implements. The class loads outputs from VPC in a previous phase and implements the `Vpc` interface on top of those outputs.

LIMITER

So far we haven't talked about limits yet. There is a step in the `Initial Setup` state machine that requests limit increases according to the desired limits in the configuration file. The step saves the current limits to the `limits` key in the DynamoDB table `ASEA-Parameters`. The `apps/app.ts` file loads the limits and passes them as an input to the phase deployment.

The `Limiter` class helps keeps track of resource we create and prevents exceeding these limits.

```
for (const { ouKey, accountKey, vpcConfig, deployments } of acceleratorConfig.getVpcConfigs()) {
  if (!limiter.create(accountKey, Limit.VpcPerRegion, region)) {
    console.log(`Skipping VPC "${vpcConfig.name}" deployment.`);
    console.log(`Reached maximum VPCs per region for account "${accountKey}" and region "${region}"`);
    continue;
  }

  createVpc({ ouKey, accountKey, vpcConfig });
}
```

Action Item: This functionality could be redesigned to scan all the constructs in a `cdk.App` and remove resource that are exceeding any limits.

Creating Stack Outputs

Initially we would create stack outputs like this:

```
new cdk.CfnOutput(stack, 'BucketOutput', {
  value: bucket.bucketArn,
});
```

But then we'd get a lot of outputs in a stack. We started some outputs together using JSON. This allowed us to store structured data inside the stack outputs.

```
new JsonOutputValue(stack, 'Output', {
  type: 'FirewallInstanceOutput',
  value: {
    instanceId: instance.instanceId,
    name: firewallConfig.name,
    az,
  },
});
```

Using the solution above, we'd not have type checking when reading or writing outputs. That's what the class `StructuredOutputValue` has a solution for. It uses the `io-ts` library to serialize and deserialize structured types.

```
export const FirewallInstanceOutput = t.interface(
  {
    id: t.string,
    name: t.string,
    az: t.string,
  },
  'FirewallInstanceOutput',
);

export type FirewallInstanceOutput = t.TypeOf<typeof FirewallInstanceOutput>;

new StructuredOutputValue<FirewallInstanceOutput>(stack, 'Output', {
  type: FirewallInstanceOutput,
  value: {
    instanceId: instance.instanceId,
    name: firewallConfig.name,
    az,
  },
});
```

And we can even improve on this a bit more.

```
export const CfnFirewallInstanceOutput = createCfnStructuredOutput(FirewallInstanceOutput);

new CfnFirewallInstanceOutput(stack, 'Output', {
  vpcId: vpc.ref,
  vpcName: vpcConfig.name,
});
```

```
export const FirewallInstanceOutputFinder = createStructuredOutputFinder(FirewallInstanceOutput, () => ({}));

// Create an OutputFinder
const firewallInstances = FirewallInstanceOutputFinder.findAll({
  outputs,
  accountKey,
});

// Example usage of the OutputFinder
const firewallInstance = firewallInstances.find(i => i.name === target.name && i.az === target.az);
```

Generally you would place the output type definition inside `src/lib/common-outputs` along with the output finder. Then in the deployment folder in `src/deployments/cdk/deployments` you would create an `output.ts` file where you would define the CDK output type with `createCfnStructuredOutput`. You would not define the CDK output type in `src/lib/common-outputs` since that project is also used by runtime code that does not need to know about CDK and CloudFormation.

ADDING TAGS TO SHARED RESOURCES IN DESTINATION ACCOUNT

There is another special type of output, `AddTagsToResourcesOutput`. It can be used to attach tags to resources that are shared into another account.

```
new AddTagsToResourcesOutput(this, 'OutputSharedResourcesSubnets', {
  dependencies: sharedSubnets.map(o => o.subnet),
  produceResources: () =>
    sharedSubnets.map(o => ({
      resourceId: o.subnet.ref,
      resourceType: 'subnet',
      sourceAccountId: o.sourceAccountId,
      targetAccountIds: o.targetAccountIds,
      tags: o.subnet.tags.renderTags(),
    })),
});
```

This will add the outputs to the stack in the account that is initiating the resource share.

Next, the state machine step `Add Tags to Shared Resources` looks for all those outputs. The step will assume the `PipelineRole` in the `targetAccountIds` and attach the given tags to the shared resource.

Custom Resources

There are different ways to create a custom resource using CDK. See the Custom Resource section for more information.

All custom resources have a `README.md` that demonstrates their usage.

EXTERNALIZING `aws-sdk`

Some custom resources set the `aws-sdk` as external dependency and some do not.

Example of setting `aws-sdk` as external dependency.

`src/lib/custom-resources/cdk-kms-grant/runtime/package.json`

```
{
  "externals": ["aws-lambda", "aws-sdk"],
  "dependencies": {
    "aws-lambda": "1.0.6",
    "aws-sdk": "2.631.0"
  }
}
```

Example of setting `aws-sdk` as embedded dependency.

`src/lib/custom-resources/cdk-guardduty-enable-admin/runtime/package.json`

```
{
  "externals": ["aws-lambda"],
  "dependencies": {
    "aws-lambda": "1.0.6",
    "aws-sdk": "2.711.0"
  }
}
```

Setting the `aws-sdk` library as external is sometimes necessary when a newer `aws-sdk` version is necessary for the Lambda runtime code. At the time of writing the NodeJS 12 runtime uses `aws-sdk` version `2.631.0`

For example the method `AWS.GuardDuty.enableOrganizationAdminAccount` was only introduced in `aws-sdk` version `2.660`. That means that Webpack has to embed the `aws-sdk` version specified in `package.json` into the compiled JavaScript file. This can be achieved by removing `aws-sdk` from the `external` array.

`src/lib/custom-resources/cdk-kms-grant/runtime/package.json`

CFN-RESPONSE

This library helps you send a custom resource response to CloudFormation.

`src/lib/custom-resources/cdk-kms-grant/runtime/src/index.ts`

```
export const handler = errorHandler(onEvent);

async function onEvent(event: CloudFormationCustomResourceEvent) {
  console.log(`Creating KMS grant...`);
  console.log(JSON.stringify(event, null, 2));

  // eslint-disable-next-line default-case
  switch (event.RequestType) {
    case 'Create':
      return onCreate(event);
    case 'Update':
      return onUpdate(event);
    case 'Delete':
      return onDelete(event);
  }
}
```

CFN-TAGS

This library helps you send attaching tags to resource created in a custom resource.

WEBPACK-BASE

This library defines the base Webpack template to compile custom resource runtime code.

`src/lib/custom-resources/cdk-kms-grant/runtime/package.json`

```
{
  "name": "@aws-accelerator/custom-resource-kms-grant-runtime",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "prepare": "webpack-cli --config webpack.config.ts"
  },
  "source": "src/index.ts",
  "main": "dist/index.js",
  "types": "dist/index.d.ts",
  "externals": ["aws-lambda", "aws-sdk"],
  "devDependencies": {
    "@aws-accelerator/custom-resource-runtime-webpack-base": "workspace:^0.0.1",
    "@types/aws-lambda": "8.10.46",
    "@types/node": "14.14.31",
    "ts-loader": "7.0.5",
    "typescript": "3.8.3",
    "webpack": "4.42.1",
    "webpack-cli": "3.3.11"
  },
  "dependencies": {
    "@aws-accelerator/custom-resource-runtime-cfn-response": "workspace:^0.0.1",
    "aws-lambda": "1.0.6",
    "aws-sdk": "2.668.0"
  }
}
```

`src/lib/custom-resources/cdk-ec2-image-finder/runtime/webpack.config.ts`

```
import { webpackConfigurationForPackage } from '@aws-accelerator/custom-resource-runtime-webpack-base';
import pkg from './package.json';

export default webpackConfigurationForPackage(pkg);
```

6.2.7 Workarounds

Stacks with Same Name in Different Regions

The reason we're creating a `cdk.App` per account and per region and per phase is because stack names across environments might overlap, and at the time of writing, the CDK CLI does not handle stacks with the same name well. For example, when there is a stack `Phase1` in `us-east-1` and another stack `Phase1` in `ca-central-1`, the stacks will both be synthesized by CDK to the `cdk.out/Phase1.template.json` file and one stack will overwrite another's output. Using multiple `cdk.App`s overcomes this issues as a different `outdir` can be set on each `cdk.App`. These `cdk.App`s are managed by the `AccountStacks` abstraction.

6.2.8 Local Development

Local Installer Stack

Use CDK to synthesize the CloudFormation template.

```
cd src/installer/cdk
pnpx cdk synth
```

The installer template file is now in `cdk.out/AcceleratorInstaller.template.json`. This file can be used to install the installer stack.

You can also deploy the installer stack directly from the command line but then you'd have to pass some stack parameters. See CDK documentation: Deploying with parameters.

```
cd accelerator/installer
pnpx cdk deploy --parameters GithubBranch=main --parameters ConfigS3Bucket=ASEA-myconfigbucket
```

Local Initial Setup Stack

There is a script called `cdk.sh` in `src/core/cdk` that allows you to deploy the Initial Setup stack.

The script sets the required environment variables and makes sure all workspace projects are built before deploying the CDK stack.

Phase Stacks

There is a script called `cdk.sh` in `src/deployments/cdk` that allows you to deploy a phase stack straight from the command-line without having to deploy the Initial Setup stack first.

The script enables development mode which means that accounts, organizations, configuration, limits and outputs will be loaded from the local environment instead of loading the values from DynamoDB. The local files that need to be available in the `src/deployments/cdk` folder are the following.

1. `accounts.json` based on `accelerator/accounts` (-Parameters table)

```
[
  {
    "key": "shared-network",
    "id": "000000000001",
    "arn": "arn:aws:organizations::000000000000:account/o-0123456789/000000000001",
    "name": "myacct-ASEA-shared-network",
    "email": "myacct+ASEA-mandatory-shared-network@example.com",
    "ou": "core"
  },
  {
    "key": "operations",
    "id": "000000000002",
    "arn": "arn:aws:organizations::000000000000:account/o-0123456789/000000000002",
    "name": "myacct-ASEA-operations",
    "email": "myacct+ASEA-mandatory-operations@example.com",
    "ou": "core"
  }
]
```

1. `organizations.json` based on `accelerator/organizations` (-Parameters table)

```
[
  {
    "ouId": "ou-0000-00000000",
    "ouArn": "arn:aws:organizations::000000000000:ou/o-0123456789/ou-0000-00000000",
    "ouName": "core",
    "ouPath": "core"
  },
  {
    "ouId": "ou-0000-00000001",
    "ouArn": "arn:aws:organizations::000000000000:ou/o-0123456789/ou-0000-00000001",
    "ouName": "prod",
    "ouPath": "prod"
  }
]
```

1. `limits.json` based on `accelerator/limits` (-Parameters table)

```
[
  {
    "accountKey": "shared-network",
    "limitKey": "Amazon VPC/VPCs per Region",
    "serviceCode": "vpc",
    "quotaCode": "L-F678F1CE",
    "value": 15,
    "region": "ca-central-1"
  },
  {
    "accountKey": "shared-network",
    "limitKey": "Amazon VPC/Interface VPC endpoints per VPC",
    "serviceCode": "vpc",
    "quotaCode": "L-29B6F2EB",
    "value": 50,
    "region": "ca-central-1"
  }
]
```

1. `outputs.json` based on the -Outputs table

```
[
  {
    "accountKey": "shared-network",
    "outputKey": "DefaultBucketOutputC7CE5936",
    "outputValue": "{\"type\":\"AccountBucket\",\"value\":{\"bucketArn\":\"arn:aws:s3:::ASEA-sharednetwork-phase1-cacentral1-18vq0emthri3h\",\"bucketName\":
\"ASEA-sharednetwork-phase1-cacentral1-18vq0emthri3h\",\"encryptionKeyArn\":\"arn:aws:kms:ca-central-1:0000000000001:key/d54a8acb-694c-4fc5-9afe-ca2b263cd0b3\",
\"region\":\"ca-central-1\"}}"
  }
]
```

1. `context.json` that contains the default values for values that are otherwise passed as environment variables.

```
{
  "acceleratorName": "ASEA",
  "acceleratorPrefix": "ASEA-",
  "acceleratorExecutionRoleName": "ASEA-PipelineRole",
  "defaultRegion": "ca-central-1"
}
```

1. `config.json` that contains the Accelerator configuration.

   The script also sets the default execution role to allow CDK to assume a role in subaccounts to deploy the phase stacks.

   Now that you have all the required local files you can deploy the phase stacks using `cdk.sh`.

```
cd src/deployments/cdk
./cdk.sh deploy --phase 1                           # deploy all phase 1 stacks
./cdk.sh deploy --phase 1 --parallel                # deploy all phase 1 stacks in parallel
./cdk.sh deploy --phase 1 --account shared-network  # deploy phase 1 stacks for account shared-network in all regions
./cdk.sh deploy --phase 1 --region ca-central-1     # deploy phase 1 stacks for region ca-central-1 for all accounts
./cdk.sh deploy --phase 1 --account shared-network --region ca-central-1 # deploy phase 1 stacks for account shared-network and region ca-central
```

   Other CDK commands are also available.

```
cd src/deployments/cdk
./cdk.sh bootstrap --phase 1
./cdk.sh synth --phase 1
```

6.2.9 Testing

We use `jest` for unit testing. There are no integration tests but this could be set-up by configuring the `Installer` CodePipeline to have a webhook on the repository and deploying changes automatically.

To run unit tests locally you can run the following command in the monorepo.

```
pnpx recursive run test -- --pass-with-no-tests --silent
```

See CDK's documentation on Testing constructs for more information on how to tests CDK constructs.

Validating Immutable Property Changes and Logical ID Changes

The most important unit test in this project is one that validates that logical IDs and immutable properties do not change unexpectedly. To avoid the issues described in section Resource Names and Logical IDs, Changing Logical IDs and Changing (Immutable) Properties.

This test can be found in the `src/deployments/cdk/test/apps/unsupported-changes.spec.ts` file. It synthesizes the `Phase` stacks using mocked outputs and uses `jest` snapshots to compare against future changes.

The test will fail when changing immutable properties or changing logical IDs of existing resources. In case the changes are expected then the snapshots will need to be updated. You can update the snapshots by running the following command.

```
pnpx run test -- -u
```

See Accept Unit Test Snapshot Changes.

Upgrade CDK

There's a test in the file `src/deployments/cdk/test/apps/unsupported-changes.spec.ts` that is currently commented out. The test takes a snapshot of the whole `Phase` stack and compares the snapshot to changes in the code.

```
test('templates should stay exactly the same', () => {
  for (const [stackName, resources] of Object.entries(stackResources)) {
    // Compare the relevant properties to the snapshot
    expect(resources).toMatchSnapshot(stackName);
  }
});
```

Before upgrading CDK we uncomment this test. We run the test to update all the snapshots. Then we update all CDK versions and run the test again to compare the snapshots with the code using the new CDK version. If the test passes, then the upgrade should be stable.

Action Item: Automate this process.

6.3 Technology Stack

We use TypeScript, NodeJS, CDK and CloudFormation. You can find some more information in the sections below.

6.3.1 TypeScript and NodeJS

In the following sections we describe the tools and libraries used along with TypeScript.

pnpm

We use the `pnpm` package manager along with `pnpm workspaces` to manage all the packages in this monorepo.

https://pnpm.js.org

https://pnpm.js.org/en/workspaces

The binary `pnpx` runs binaries that belong to `pnpm` packages in the workspace.

https://pnpm.js.org/en/pnpx-cli

prettier

We use `prettier` to format code in this repository. A GitHub action makes sure that all the code in a pull requests adheres to the configured `prettier` rules. See Github Actions.

eslint

We use `eslint` as a static analysis tool that checks our TypeScript code. A GitHub action makes sure that all the code in a pull requests adheres to the configured `eslint` rules. See Github Actions.

6.3.2 CloudFormation

CloudFormation deploys both the Accelerator stacks and resources and the deployed stacks and resources. See Operations Guide: System Overview for the distinction between Accelerator resources and deployed resources.

6.3.3 CDK

AWS CDK defines the cloud resources in a familiar programming language. While AWS CDK supports TypeScript, JavaScript, Python, Java, and C#/.Net, the contributions should be made in Typescript, as outlined in the Accelerator Development First Principles.

Developers can use programming languages to define reusable cloud components known as Constructs. You compose these together into Stacks and Apps. Learn more at https://docs.aws.amazon.com/cdk/latest/guide/home.html

6.4 Best Practices

6.4.1 TypeScript and NodeJS

Handle Unhandled Promises

Entry point TypeScript files -- files that start execution instead of just defining methods and classes -- should have the following code snippet at the start of the file.

```
process.on('unhandledRejection', (reason, _) => {
  console.error(reason);
  process.exit(1);
});
```

This prevents unhandled promise rejection errors by NodeJS. Please read https://medium.com/dailyjs/how-to-prevent-your-node-js-process-from-crashing-5d40247b8ab2 for more information.

6.4.2 CloudFormation

Cross-Account/Region References

When managing multiple AWS accounts, the Accelerator may need permissions to modify resources in the managed accounts. For example, a transit gateway could be created in a shared network account and it need to be shared to the perimeter account to create a VPN connection.

In a single-account environment we would could just:

1. create a single stack and use `!Ref` to refer to the transit gateway;
2. or deploy two stacks
3. one stack that contains the transit gateway and creates a CloudFormation exported output that contains the transit gateway ID;
4. another stack that imports the exported output value from the previous stack and uses it to create a VPN connection.

In a multi-account environment this is not possible and we had to find a way to share outputs across accounts and regions.

See Passing Outputs Between Phases.

Resource Names and Logical IDs

Some resources, like `AWS::S3::Bucket`, can have an explicit name. Setting an explicit name can introduce some possible issues.

The first issue that could occur goes as follows:

- the named resource has a retention policy to retain the resource after deleting;
- then the named resource is created through a CloudFormation stack;
- next, an error happens while creating or updating the stack and the stack rolls back;
- and finally the named resource is deleted from the stack but has a retention policy to retain, so the resource not be deleted;

Suppose then that the stack creation issue is resolved and we retry to create the named resource through the CloudFormation stack:

- the named resource is created through a CloudFormation stack;
- the named resource will fail to create because a resource with the given name already exists.

The best way to prevent this issue from happening is to not explicitly set a name for the resource and let CloudFormation generate the name.

Another issue could occur when changing the logical ID of the named resource. This is documented in the following section.

Changing Logical IDs

When changing the logical ID of a resource CloudFormation assumes the resource is a new resource since it has a logical ID it does not know yet. When updating a stack, CloudFormation will always prioritize resource creation before deletion.

The following issue could occur when the resource has an explicit name. CloudFormation will try to create the resource anew and will fail since a resource with the given name already exists. Example of resources where this could happen are `AWS::S3::Bucket`, `AWS::SecretManager::Secret`.

Changing (Immutable) Properties

Not only changing logical IDs could cause CloudFormation to replace resources. Changing immutable properties also cause replacement of resources. See Update behaviors of stack resources.

Be especially careful when:

- changing immutable properties for a named resource. Example of a resource is `AWS::Budgets::Budget`, `AWS::ElasticLoadBalancingV2::LoadBalancer`.
- updating network interfaces for an `AWS::EC2::Instance`. Not only will this cause the instance to re-create, it will also fail to attach the network interfaces to the new EC2 instance. CloudFormation creates the new EC2 instance first before deleting the old one. It will try to attach the network interfaces to the new instance, but the network interfaces are still attached to the old instance and CloudFormation will fail.

For some named resources, like `AWS::AutoScaling::LaunchConfiguration` and `AWS::Budgets::Budget`, we append a hash to the name of the resource that is based on its properties. This way when an immutable property is changed, the name will also change, and the resource will be replaced successfully. See for example `src/lib/cdk-constructs/src/autoscaling/launch-configuration.ts` and `src/lib/cdk-constructs/src//billing/budget.ts`.

```
export type LaunchConfigurationProps = autoscaling.CfnLaunchConfigurationProps;

/**
 * Wrapper around CfnLaunchConfiguration. The construct adds a hash to the launch configuration name that is based on
 * the launch configuration properties. The hash makes sure the launch configuration gets replaced correctly by
 * CloudFormation.
 */
export class LaunchConfiguration extends autoscaling.CfnLaunchConfiguration {
  constructor(scope: cdk.Construct, id: string, props: LaunchConfigurationProps) {
    super(scope, id, props);

    if (props.launchConfigurationName) {
      const hash = hashSum({ ...props, path: this.node.path });
      this.launchConfigurationName = `${props.launchConfigurationName}-${hash}`;
    }
  }
}
```

6.4.3 CDK

CDK makes heavy use of CloudFormation so all best practices that apply to CloudFormation also apply to CDK.

Logical IDs

The logical ID of a CDK component is calculated based on its path in the construct tree. Be careful moving around constructs in the construct tree -- e.g. changing the parent of a construct or nesting a construct in another construct -- as this will change the logical ID of the construct. Then you could end up with the issues described in section Changing Logical IDs and section Changing (Immutable) Properties.

See Logical ID Stability for more information.

Moving Resources between Nested Stacks

In some cases we use nested stacks to overcome the limit of 200 CloudFormation resources per stack.

In the code snippet below you can see how we generate a dynamic amount of nested stack based on the amount of interface endpoints we construct. The `InterfaceEndpoint` construct contains CloudFormation resources so we have to be careful to not exceed the limit of 200 CloudFormation resources per nested stack. That is why we limit the amount of interface endpoints to 30 per nested stack.

```
let endpointCount = 0;
let endpointStackIndex = 0;
let endpointStack;
for (const endpoint of endpointConfig.endpoints) {
  if (!endpointStack || endpointCount >= 30) {
    endpointStack = new NestedStack(accountStack, `Endpoint${endpointStackIndex++}`);
    endpointCount = 0;
  }
  new InterfaceEndpoint(endpointStack, pascalCase(endpoint), {
    serviceName: endpoint,
  });
  endpointCount++;
}
```

We have to be careful here though. Suppose the configuration file contains 40 interface endpoints. The first 30 interface endpoints will be created in the first nested stack; the next 10 interface endpoints will be created in the second nested stack. Suppose now that we remove the first nested endpoint from the configuration file. This will cause the 31st interface endpoint to become the 30th interface endpoint in the list and it will cause the interface endpoint to be moved from the second nested stack to the first nested stack. This will cause the stack updates to fail since CloudFormation will first try to create the interface endpoint in the first nested stack before removing it from the second nested stack. We do currently not support changes to the interface endpoint configuration because of this behavior.

L1 vs. L2 Constructs

See AWS Construct library for an explanation on L1 and L2 constructs.

The L2 constructs for EC2 and VPC do not map well onto the Accelerator-managed resources. For this reason we mostly use L1 CDK constructs -- such as `ec2.CfnVPC`, `ec2.CfnSubnet` -- instead of using L2 CDK constructs -- such as `ec2.Vpc` and `ec2.Subnet`.

CDK Code Dependency on Lambda Function Code

You can read about the distinction between CDK code and runtime code in the introduction of the Development section.

CDK code can depend on runtime code. For example when we want to create a Lambda function using CDK, we need the runtime code to define the Lambda function. We use `npm scripts`, `npm` dependencies and the `NodeJS` `modules` API to define this dependency between CDK code and runtime code.

First of all, we create a separate folder that contains the workspace and runtime code for our Lambda function. Throughout the project we've called these workspaces `...-lambda` but it could also be named `...-runtime`. See `src/lib/custom-resources/cdk-acm-import-certificate/runtime/package.json`.

This workspace's `package.json` file needs a `prepare` script that compiles the runtime code. See `npm-scripts`.

The `package.json` file also needs a `name` and a `main` entry that points to the compiled code.

`runtime/package.json`

```
{
  "name": "lambda-fn-runtime",
  "main": "dist/index.js",
  "scripts": {
    "prepare": "webpack-cli --config webpack.config.ts"
  }
}
```

Now when another workspace depends on our Lambda function runtime code workspace, the `prepare` script will run and it will compile the Lambda function runtime code.

Next, we add the dependency to the new workspace to the workspace that contains the CDK code using `pnpm` or by adding it to `package.json`.

`cdk/package.json`

```
{
  "devDependencies": {
    "lambda-fn-runtime": "workspace:^0.0.1"
  }
}
```

In the CDK code we can now resolve the path to the compiled code using the `NodeJS` `modules` API. See NodeJS `modules` API.

`cdk/src/index.ts`

```
class LambdaFun extends cdk.Construct {
  constructor(scope: cdk.Construct, id: string) {
    super(scope, id);

    // Find the runtime package folder and resolves the `main` entry of `package.json`.
    // In our case this is `node_modules/lambda-fn-runtime/dist/index.js`.
    const runtimeMain = resolve.require('lambda-fn-runtime');

    // Find the directory containing our `index.js` file.
    // In our case this is `node_modules/lambda-fn-runtime/dist`.
    const runtimeDir = path.dirname(lambdaPath);

    new lambda.Function(this, 'Resource', {
      runtime: lambda.Runtime.NODEJS_14_X,
      code: lambda.Code.fromAsset(runtimeDir),
      handler: 'index.handler', // The `handler` function in `index.js`
    });
  }
}
```

You now have a CDK Lambda function that uses the compiled Lambda function runtime code.

Note: The runtime code needs to recompile every time it changes since the `prepare` script only runs when the runtime workspace is installed.

Custom Resource

We create custom resources for functionality that is not supported natively by CloudFormation. We have two types of custom resources in this project:

1. Custom resource that calls an SDK method;
2. Custom resource that needs additional functionality and is backed by a custom Lambda function.

CDK has a helper construct for the first type of custom resources. See CDK `AwsCustomResource` documentation. This helper construct is for example used in the custom resource `ds-log-subscription`.

The second type of custom resources requires a custom Lambda function runtime as described in the previous section. For example `acm-import-certificate` is backed by a custom Lambda function.

Only a single Lambda function is created per custom resource, account and region. This is achieved by creating only a single Lambda function in the construct tree.

`src/lib/custom-resources/custom-resource/cdk/index.ts`

```
class CustomResource extends cdk.Construct {
  constructor(scope: cdk.Construct, id: string, props: CustomResourceProps) {
    super(scope, id);

    new cdk.CustomResource(this, 'Resource', {
      resourceType: 'Custom::CustomResource',
      serviceToken: this.lambdaFunction.functionArn,
    });
  }

  private get lambdaFunction() {
    const constructName = `CustomResourceLambda`;

    const stack = cdk.Stack.of(this);
    const existing = stack.node.tryFindChild(constructName);
    if (existing) {
      return existing as lambda.Function;
    }

    // The package '@aws-accelerator/custom-resources/cdk-custom-resource-runtime' contains the runtime code for the custom resource
    const lambdaPath = require.resolve('@aws-accelerator/custom-resources/cdk-custom-resource-runtime');
    const lambdaDir = path.dirname(lambdaPath);

    return new lambda.Function(stack, constructName, {
      code: lambda.Code.fromAsset(lambdaDir),
    });
  }
}
```

Escape Hatches

Sometimes CDK does not support a property on a resource that CloudFormation does support. You can then override the property using the `addOverride` or `addPropertyOverride` methods on CDK CloudFormation resources. See CDK escape hatches.

AUTOSCALING GROUP METADATA

An example where we override metadata is when we create a launch configuration.S

```
const launchConfig = new autoscaling.CfnLaunchConfiguration(this, 'LaunchConfig', { ... });

launchConfig.addOverride('Metadata.AWS::CloudFormation::Authentication', {
  S3AccessCreds: {
    type: 'S3',
    roleName,
    buckets: [bucketName],
  },
});

launchConfig.addOverride('Metadata.AWS::CloudFormation::Init', {
  configSets: {
    config: ['setup'],
  },
  setup: {
    files: {
      // Add files here
    },
    services: {
      // Add services here
    },
    commands: {
      // Add commands here
    },
  },
});
```

SECRET `SECRETVALUE`

Another example is when we want to use `secretsmanager.Secret` and set the secret value.

```
function setSecretValue(secret: secrets.Secret, value: string) {
  const cfnSecret = secret.node.defaultChild as secrets.CfnSecret; // Get the L1 resource that backs this L2 resource
  cfnSecret.addPropertyOverride('SecretString', value); // Override the property `SecretString` on the L1 resource
  cfnSecret.addPropertyDeletionOverride('GenerateSecretString'); // Delete the property `GenerateSecretString` from the L1 resource
}
```

6.5 How to Contribute

6.5.1 How-to

6.5.2 Adding New Functionality?

Before making a change or adding new functionality you have to verify what kind of functionality is being added.

- Is it an Accelerator-management change?
- Is the change related to the `Installer` stack?
- Is the change CDK related?
- Make the change in `src/installer/cdk`.
- Is the change runtime related?
- Make the change in `src/installer/cdk/assets`.
- Is the change related to the `Initial Setup` stack?
- Is the change CDK related?
- Make the change in `src/core/cdk`
- Is the change runtime related?
- Make the change in `src/core/runtime`
- Is it an Accelerator-managed change?
- Is the change related to the `Phase` stacks?
- Is the change CDK related?
- Make the change in `src/deployments/cdk`
- Is the change runtime related?
- Make the change in `src/deployments/runtime`

6.5.3 Create a CDK Lambda Function with Lambda Runtime Code

See CDK Code Dependency on Lambda Function Code for a short introduction.

6.5.4 Create a Custom Resource

See Custom Resource and Custom Resources for a short introduction.

1. Create a separate folder that contains the CDK and Lambda function runtime code, e.g. `src/lib/custom-resources/my-custom-resource`;
2. Create a folder `my-custom-resource` that contains the CDK code;
3. Create a `package.json` file with a dependency to the `my-custom-resource/runtime` package;
4. Create a `cdk` folder that contains the source of the CDK code;
5. Create a folder `my-custom-resource/runtime` that contains the runtime code;
6. Create a `runtime/package.json` file with a `"name"`, `"prepare"` script and a `"main"`;
7. Create a `runtime/webpack.config.ts` file that compiles TypeScript code to a single JavaScript file;
8. Create a `runtime/src` folder that contains the source of the Lambda function runtime code;

You can look at the `src/lib/custom-resources/cdk-acm-import-certificate` custom resource as an example.

It is best practice to add tags to any resources that the custom resource creates using the `cfn-tags` library.

6.5.5 Run All Unit Tests

Run in the root of the project.

```
pnpm recursive run test --no-bail --stream -- --silent
```

6.5.6 Accept Unit Test Snapshot Changes

Run in `src/deployments/cdk`.

```
pnpm run test -- -u
```

6.5.7 Validate Code with Prettier

Run in the root of the project.

```
pnpx prettier --check **/*.ts
```

6.5.8 Format Code with Prettier

Run in the root of the project.

```
pnpx prettier --write **/*.ts
```

6.5.9 Validate Code with `tslint`

Run in the root of the project.

```
pnpm recursive run lint --stream --no-bail
```

6.6 AWS Internal - Accelerator Release Process

6.6.1 Creating a new Accelerator Code Release

1. Ensure `main` branch is in a suitable state
2. Disable branch protection for both the `main` branch and for the `release/` branches
3. Create a version branch with SemVer semantics and a `release/` prefix: e.g. `release/v1.0.5` or `release/v1.0.5-b`
4. On latest `main`, run: `git checkout -b release/vX.Y.Z`
5. **Important:** Certain git operations are ambiguous if tags and branches have the same name. Using the `release/` prefix reserves the actual version name for the tag itself; i.e. every `release/vX.Y.Z` branch will have a corresponding `vX.Y.Z` tag.
6. Push that branch to GitHub (if created locally)
7. `git push origin release/vX.Y.Z`
8. The release workflow will run, and create a **DRAFT** release if successful with all commits since the last tagged release.
9. Prune the commits that have been added to the release notes (e.g. remove any low-information commits)
10. Publish the release - this creates the git tag in the repo and marks the release as latest. It also bumps the `version` key in several project `package.json` files.
11. Re-enable branch protection for both the `main` branch and for the `release/` branches
12. Note: The `Publish` operation will run the following GitHub Action, which merges the `release/vX.Y.Z` branch to `main`. **Branch Protection in GitHub will cause this to fail**, and why we are momentarily disabling branch protection.
13. A successful run of this workflow will automatically kick off the "Generate Documentation" workflow. This workflow may also be initiated at any time manually via the GitHub Actions UI (since it is configured as a `workflow_dispatch` action).
14. once the documentation is generated, add the ZIP file to the release assets, named `AWS-SEA-Documentation-vXXX.zip`
15. Rename the `AcceleratorInstaller.template.json` to `AcceleratorInstaller`XXX`.template.json` replacing XXX with the version number without punctuation (i.e. `AcceleratorInstaller121b.template.json`).
    - Repeat for `AcceleratorInstaller-CodeCommit.template-vXXX.json`.
16. Add the Accelerator configuration file schema documentation ZIP to the release assets, named `AWS-SEA-Config-Schema-vXXX-DRAFT.zip`.
17. Add the Accelerator ***Alpha*** GUI ZIP to the release assets, named `AWS-SEA-GUI-mockup-DoNotUse-vXXX-alpha.zip`.

...Return to Accelerator Table of Contents

7. Architectures

7.1 PBMM

7.1.1 AWS Secure Environment Accelerator PBMM Architecture

The *AWS Secure Environment PBMM Architecture* is a comprehensive, multi-account AWS cloud architecture, initially designed for use within the Government of Canada for PBMM workloads. The *AWS Secure Environment PBMM Architecture* has been designed to address central identity and access management, governance, data security, comprehensive logging, and network design/segmentation per Canadian Centre for Cyber Security ITSG-33 specifications (a NIST 800-53 variant).

This document specifically does NOT talk about the tooling or mechanisms that can be used to deploy the architecture. While the AWS Secure Environment Accelerator (SEA) is one tool capable of deploying this architecture (along with many other architectures), customers can use whatever mechanism they deem appropriate to deploy it. This document does not discuss the AWS SEA tooling or architecture and is strictly focused on the resulting deployed solution created by using the provided sample PBMM Accelerator configuration file. This architecture document should stand on its own in depicting the `deployed` architecture. Users looking for information on the SEA tooling itself, should refer to the other SEA documents.

It is anticipated we will offer multiple sample architectures with the AWS SEA solution, each having its own architecture document like this. As the SEA can produce hundreds of solutions, it does not make sense to repeat that content in this document.

Introduction

The *AWS Secure Environment Architecture* has been built with the following design principles:

1. Maximize agility, scalability, and availability
2. Enable the full capability of the AWS cloud
3. Be adaptable to evolving technological capabilities in the underlying platform being used in the *AWS Secure Environment Architecture*
4. Allow for seamless auto-scaling and provide unbounded bandwidth as bandwidth requirements increase (or decrease) based on actual customer load (a key aspect of the value proposition of cloud computing)
5. Design for high availability: the design stretches across two physical AWS Availability Zones (AZ), such that the loss of any one AZ does not impact application availability. The design can be easily extended to a third availability zone.
6. Operate as least privilege: all principals in the accounts are intended to operate with the lowest-feasible permission set.

PURPOSE OF DOCUMENT

This document is intended to outline the technical measures that are delivered by the *AWS Secure Environment Architecture* that make it suitable for PBMM workloads. An explicit **non-goal** of this document is to explain the delivery architecture of the AWS Secure Environment Accelerator tool itself, an open-source software project built by AWS.

While the central purpose of the AWS Secure Environment Accelerator is to establish an *AWS Secure Environment Architecture* into an AWS account footprint, this amounts to an implementation detail as far as the *AWS Secure Environment Architecture* is concerned. The *AWS Secure Environment Architecture* is a standalone design, irrespective of how it was delivered into a customer AWS environment. It is nonetheless anticipated that most customers will choose to realize their *AWS Secure Environment Architecture* via the delivery mechanism of the AWS Secure Environment Accelerator tool.

Comprehensive details on the tool itself are available elsewhere:

1. AWS Secure Environment Accelerator tool Operations & Troubleshooting Guide
2. AWS Secure Environment Accelerator tool Developer Guide

Except where absolutely necessary, this document will refrain from referencing the *AWS Secure Environment Accelerator tool* further.

OVERVIEW

The central features of the *AWS Secure Environment Architecture* are as follows:

- **AWS Organization with multiple-accounts:** An AWS Organization is a grouping construct for a number of separate AWS accounts that are controlled by a single customer entity. This provides consolidated billing, organizational units, and facilitates the deployment of pan-Organizational guardrails such as CloudTrail logs and Service Control Policies. The separate accounts provide strong control-plane and data-plane isolation between workloads and/or environments.
- **Encryption:** AWS KMS with customer-managed CMKs is used extensively for any data stored at rest, in S3 buckets, EBS volumes, RDS encryption.
- **Service Control Policies:** SCPs provide a guardrail mechanism principally used to deny entire categories of API operations at an AWS account, OU, or Organization level. These can be used to ensure workloads are deployed only in prescribed regions, ensure only whitelisted services are used, or prevent the disablement of detective/preventative controls. Prescriptive SCPs are provided.
- **Centralized, Isolated Networking:** Virtual Private Clouds (VPCs) are used to create data-plane isolation between workloads, centralized in a shared-network account. Connectivity to on-prem environments, internet egress, shared resources and AWS APIs are mediated at a central point of ingress/egress via the use of Transit Gateway, Site-to-Site VPN, Next-Gen Firewalls, and AWS Direct Connect (where applicable).
- **Centralized DNS Management:** Amazon Route 53 is used to provide unified public and private hosted zones across the cloud environment. Inbound and Outbound Route 53 Resolvers extend this unified view of DNS to on-premises networks.
- **Comprehensive Logging:** CloudTrail logs are enabled Organization-wide to provide auditability across the cloud environment. CloudWatch Logs, for applications, as well as VPC flow logs, are centralized and deletion is prevented via SCPs.
- **Detective Security Controls:** Potential security threats are surfaced across the cloud environment via automatic deployment of detective security controls such as GuardDuty, AWS Config, and Security Hub.
- **Single-Sign-On**: AWS SSO is used to provide AD-authenticated IAM role assumption into accounts across the Organization for authorized principals.

DOCUMENT CONVENTION

Several conventions are used throughout this document to aid understanding.

AWS Account Numbers

AWS account numbers are decimal-digit pseudorandom identifiers with 12 digits (e.g. `651278770121`). This document will use the convention that an AWS Organization Management (root) account has the account ID `123456789012`, and child accounts are given by `111111111111`, `222222222222`, etc.

For example the following ARN would refer to a VPC subnet in the `ca-central-1` region in the Organization Management (root) account:

```
arn:aws:ec2:ca-central-1:123456789012:subnet/subnet-024759b61fc305ea3
```

JSON Annotation

Throughout the document, JSON snippets may be annotated with comments (starting with `//`). The JSON language itself does not define comments as part of the specification; these must be removed prior to use in most situations, including the AWS Console and APIs.

For example:

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::123456789012:root" // Trust the Organization Management (root) account.
  },
  "Action": "sts:AssumeRole"
}
```

The above is not valid JSON without first removing the comment on the fourth line.

IP Addresses

The design makes use of RFC1918 addresses (e.g. `10.1.0.0/16`) and RFC6598 (e.g. `100.96.250.0/23`) for various networks; these will be labeled accordingly. Any specific range or IP shown is purely for illustration purposes only.

DEPARTMENT NAMING

This document will make no reference to specific Government of Canada departments. Where naming is required (e.g. in domain names), this document will use a placeholder name as needed; e.g. `dept.gc.ca`.

RELATIONSHIP TO AWS LANDING ZONE

AWS Landing Zone is an AWS Solution designed to deploy multi-account cloud architectures for customers. The *AWS Secure Environment Architecture* draws on design patterns from Landing Zone, and re-uses several concepts and nomenclature, but it is not directly derived from it. An earlier internal release of the *AWS Secure Environment Architecture* presupposed the existence of an AWS Landing Zone in the Organization; this requirement has since been removed as of release `v1.1.0`.

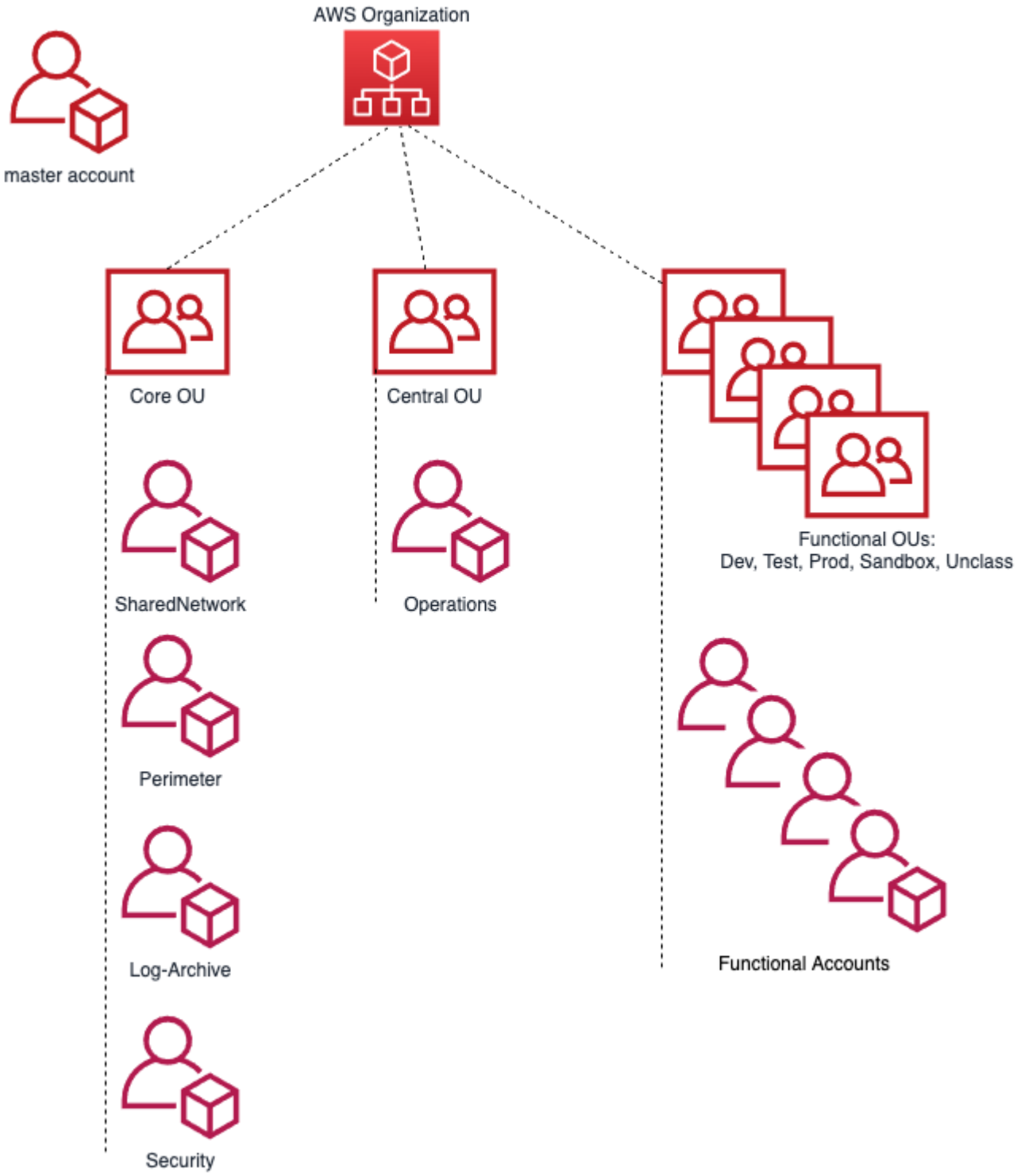
7.1.2 Account Structure

AWS accounts are a strong isolation boundary; by default there is zero control plane or data plane access from one AWS account to another. AWS Organizations is a service that provides centralized billing across a fleet of accounts, and optionally, some integration-points for cross-account guardrails and cross-account resource sharing. The *AWS Secure Environment Architecture* uses these features of AWS Organizations to realize its design.

Accounts

The *AWS Secure Environment Architecture* includes the following AWS accounts.

Note that the account structure is strictly a control plane concept - nothing about this structure implies anything about the network design or network flows.


AWS Organization
master account
Core OU
Central OU
Functional OUs:
Dev, Test, Prod, Sandbox, Unclass
SharedNetwork
Operations
Perimeter
Log-Archive
Functional Accounts
Security

Organization Management (root) AWS Account

The AWS Organization resides in the Organization Management (root) AWS account. This account is not used for workloads (to the full extent possible) - it functions primarily as a billing aggregator, and a gateway to the entire cloud footprint for a high-trust principal. There exists a trust relationship between child AWS accounts in the Organization and the Organization Management (root) account; i.e. the child accounts have a role of this form:

```
{
  "Role": {
    "Path": "/",
    "RoleName": "AWSCloudFormationStackSetExecutionRole",
    "Arn": "arn:aws:iam::111111111111:role/AWSCloudFormationStackSetExecutionRole", // Child account.
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "AWS": "arn:aws:iam::123456789012:root" // Organization Management (root) account may assume this role.
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
}
```

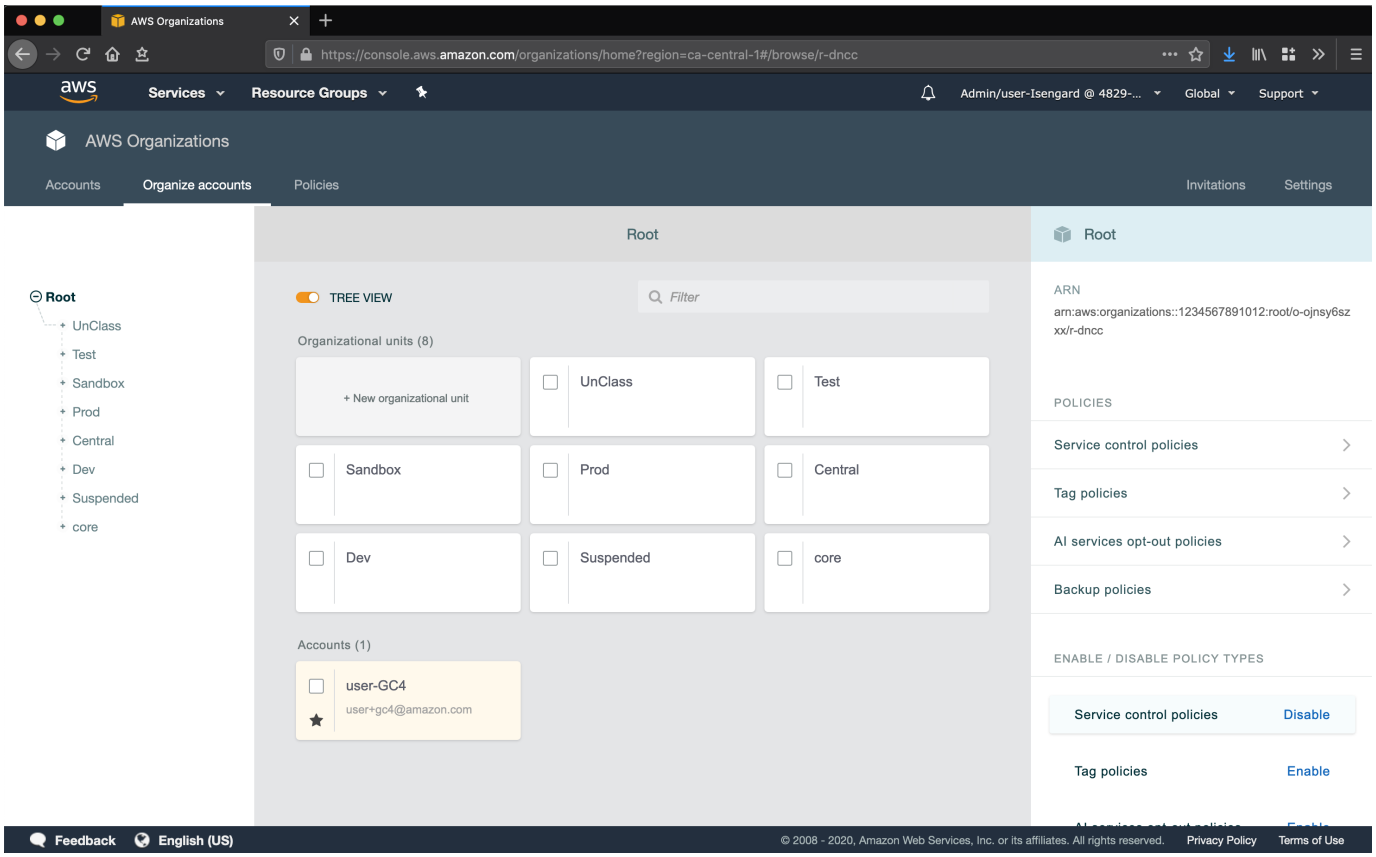
Note that this is a different role name than the default installed by AWS Organizations (`OrganizationAccountAccessRole`).

AWS SSO

AWS SSO resides in the Organization Management (root) account in the organization, due to a current requirement of the AWS SSO service. This service deploys IAM roles into the accounts in the Organization. More details on SSO are available in the **Authentication and Authorization** section.

ORGANIZATIONAL UNITS

Underneath the root of the Organization, Organizational Units (OUs) provide an optional mechanism for grouping accounts into logical collections. Aside from the benefit of the grouping itself, these collections serve as the attachment points for SCPs (preventative API-blocking controls), and Resource Access Manager sharing (cross-account resource sharing).



Example use cases are as follows:

- An SCP is attached to the core OU to prevent the deletion of Transit Gateway resources in the associated accounts.
- The shared network account uses RAM sharing to share the development line-of-business VPC with a development OU. This makes the VPC available to a functional account in that OU used by developers, despite residing logically in the shared network account.

OUs may be nested (to a total depth of five), with SCPs and RAM sharing applied at the desired level. A typical *AWS Secure Environment Architecture* environment will have the following OUs:

Core OU

This OU houses all administrative accounts, such as the core landing zone accounts. No application accounts or application workloads are intended to exist within this OU. This OU also contains the centralized networking infrastructure in the `SharedNetwork` account.

Central OU

This OU houses accounts containing centralized resources, such as a shared AWS Directory Service (Microsoft AD) instance. Other shared resources such as software development tooling (source control, testing infrastructure), or asset repositories should be created in this OU.

Functional OU: Sandbox

This OU contains a set of Sandbox accounts used by development teams for proof of concept / prototyping work. These accounts are isolated at a network level and are not connected to the VPCs hosting development, test and production workloads. These accounts have direct internet access via an internet gateway (IGW). They do not route through the Perimeter Security services VPC for internet access.

Functional OU: UnClass

Accounts in this OU host unclassified application solutions. These accounts have internet access via the Perimeter firewall. This is an appropriate place to do cross-account unclassified collaboration with other departments or entities, or test services that are not available in the Canadian region.

Functional OU: Dev

Accounts in this OU host development tools and line of business application solutions that are part of approved releases and projects. These accounts have internet access via the Perimeter firewall.

Functional OU: Test

Accounts in this OU host test tools and line of business application solutions that are part of approved releases and projects. These accounts have internet access via the Perimeter firewall.

Functional OU: Prod

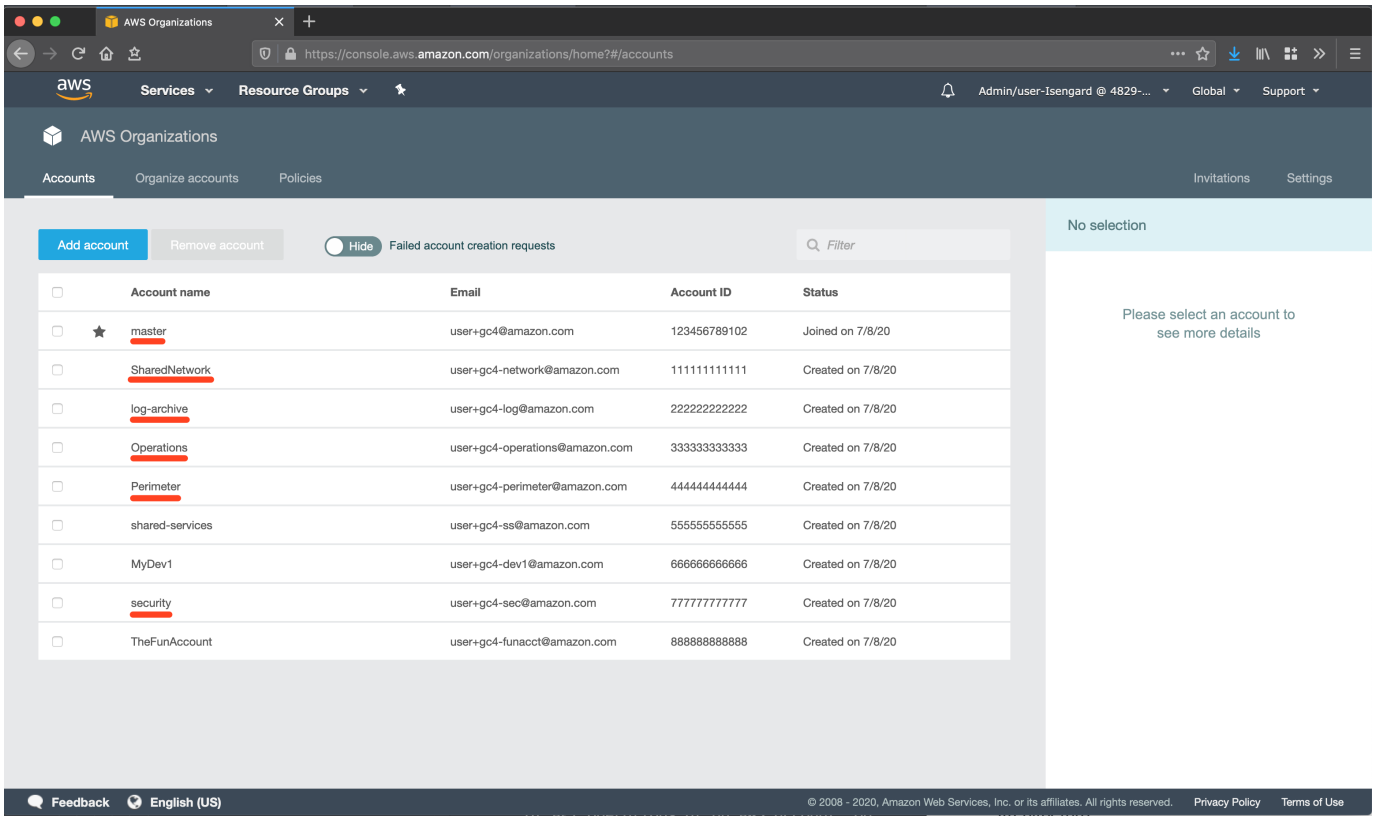
Accounts in this OU host production tools and line of business application solutions that are part of approved releases and projects. These accounts have internet access via the Perimeter firewall. Accounts in this OU are locked down with only specific Operations and Security personnel having access.

Suspended OU

A suspended OU is created to act as a container for end-of-life accounts or accounts with suspected credential leakage. The `DenyAll` SCP is applied, which prevents all control-plane API operations from taking place by any account principal.

Mandatory Accounts

The *AWS Secure Environment Architecture* is an opinionated design, which partly manifests in the accounts that are deemed mandatory within the Organization. The following accounts are assumed to exist, and each has an important function with respect to the goals of the overall Architecture (mandatory in red)



ORGANIZATION MANAGEMENT (ROOT)

As discussed above, the Organization Management (root) account functions as the root of the AWS Organization, the billing aggregator, attachment point for SCPs. Workloads are not intended to run in this account.

Note: Customers deploying the *AWS Secure Environment Architecture* via the AWS Secure Environment Accelerator will deploy into this account. See the Operations Guide for more details.

PERIMETER

The perimeter account, and in particular the perimeter VPC therein, functions as the single point of ingress/egress from the PBMM cloud environment to the public internet and/or on-premises network. This provides a central point of network control through which all workload-generated traffic, ingress and egress, must transit. The perimeter VPC hosts next-generation firewall instances that provide security services such as virus scanning, malware protection, intrusion protection, TLS inspection, and web application firewall functionality. More details on can be found in the Networking section of this document.

SHARED NETWORK

The shared network account hosts the vast majority of the AWS-side of the networking resources throughout the *AWS Secure Environment Architecture*. Workload-scoped VPCs (`Dev` , `Test` , `Prod` , etc) are defined here, and shared via RAM sharing to the respective OUs in the Organization. A Transit Gateway provides connectivity from the workloads to the internet or on-prem, without permitting cross-environment (AKA "East:West traffic") traffic (e.g. there is no Transit Gateway route from the `Dev` VPC to the `Prod` VPC). More details on can be found in the Networking section of this document.

OPERATIONS

The operations account provides a central location for the cloud team to provide cloud operation services to other AWS accounts within the Organization and is where an organizations cloud operations team "hangs out" or delivers tooling applicable across all accounts in the organization. It provides ViewOnly access to CloudWatch logs and metrics across the organization. It's where centralized Systems Manager Session Manager Automation (remediation) documents are located. It's where organizations centralize backup automation (if automated), SSM inventory and patch jobs, and where AWS Managed Active Directory would typically be deployed and accessible to all workloads in the organization. In some AWS documentation this is referred to as the Shared Services account.

LOG ARCHIVE

The log archive account provides a central aggregation and secure storage point for all audit logs created within the AWS Organization. This account contains a centralized location for copies of every account's Audit and Configuration compliance logs. It also provides a storage location for any other audit/compliance logs, as well as application/OS logs.

The AWS CloudTrail service provides a full audit history of all actions taken against AWS services, including users logging into accounts. We recommend access to this account be restricted to auditors or security teams for compliance and forensic investigations related to account activity. Additional CloudTrail trails for operational use can be created in each account.

SECURITY

The security account is restricted to authorized security and compliance personnel, and related security or audit tools. This is an aggregation point for security services, including AWS Security Hub, and serves as the Organization Management (root) for Amazon Guard Duty. A trust relationship with a readonly permission policy exists between every Organization account and the security account for audit and compliance purposes.

DEVOPS ACCOUNT AND/OR SHARED TEAM ACCOUNTS

Used to deliver CI/CD capabilities - two patterns are depicted in the architecture diagrams - The first has a single org wide central CI/CD tooling account, the other has a CI/CD and shared tooling account per major application team/grouping of teams/applications. Which is used will depend entirely on the org size, maturity model, delegation model of the organization and their team structures. We would generally still recommend CI/CD tooling in each developer account (i.e. using Code Commit) and when certain branch names were leveraged, causes the branch/PR to be automatically pulled into the centralized CI/CD tooling account and the approvals and promotion process which will push the code through the SDLC cycle to Test and Prod accounts, etc. Refer to this blog for more details on automating this pattern.

Functional Accounts

Functional accounts are created on demand, and placed into an appropriate OU in the Organization structure. The purpose of functional accounts is to provide a secure and managed environment where project teams can use AWS resources. They provide an isolated control plane so that the actions of one team in one account cannot inadvertently affect the work of other teams in other accounts.

Functional accounts will gain access to the RAM shared resources of their respective parent OU. Accounts created for `systemA` and `systemB` in the `Dev` OU would have control plane isolation from each other; however these would both have access to the `Dev` VPC (shared from the `SharedNetwork` account).

Data plane isolation within the same VPC is achieved by default, by using appropriate security groups whenever ingress is warranted. For example, the app tier of `systemA` should only permit ingress from the `systemA-web` security group, not an overly broad range such as `0.0.0.0/0`, or even the VPC range.

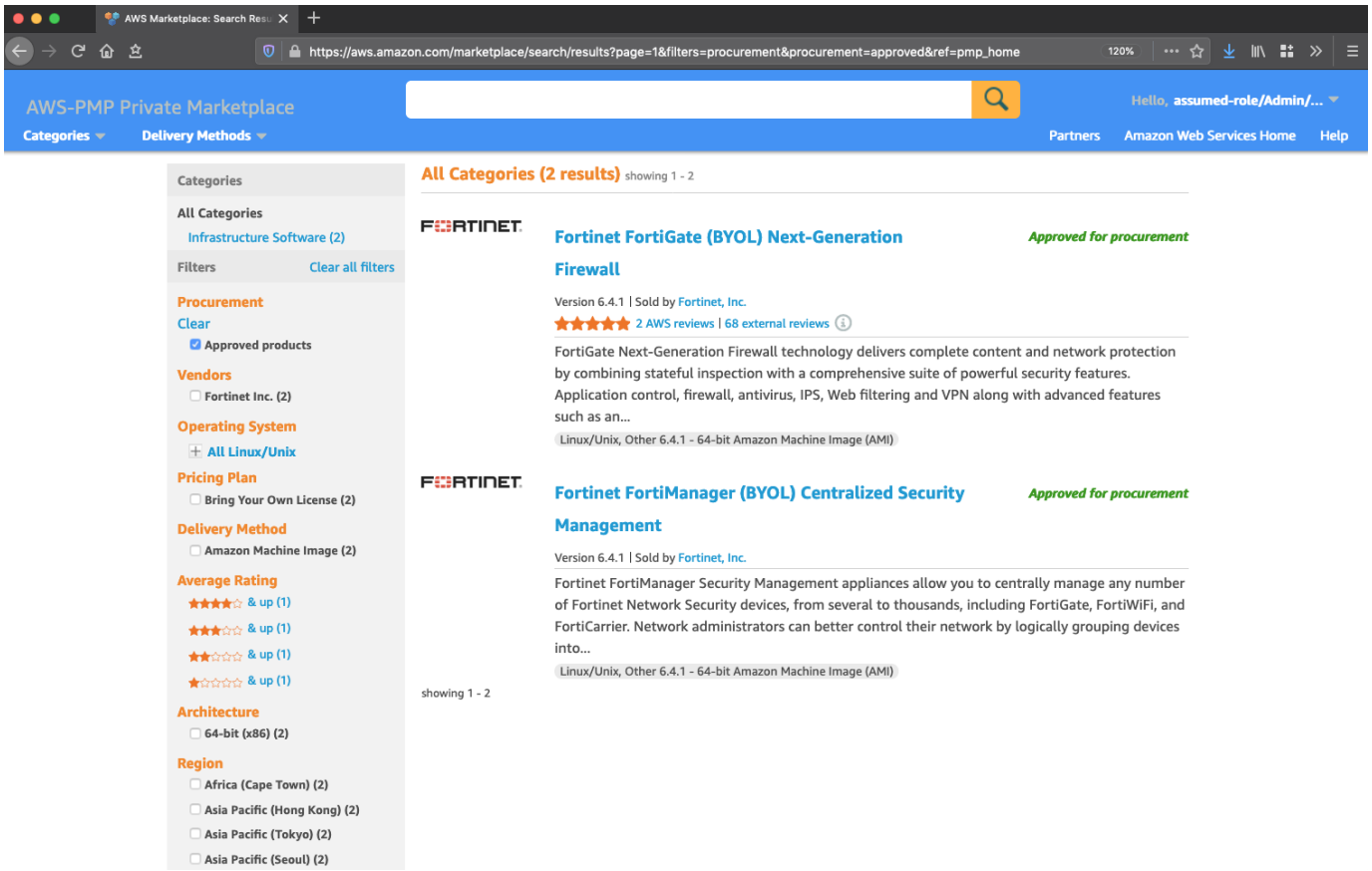
Account Level Settings

The *AWS Secure Environment Architecture* recommends the enabling of certain account-wide features on account creation. Namely, these include:

1. S3 Public Access Block
2. By-default encryption of EBS volumes.

Private Marketplace

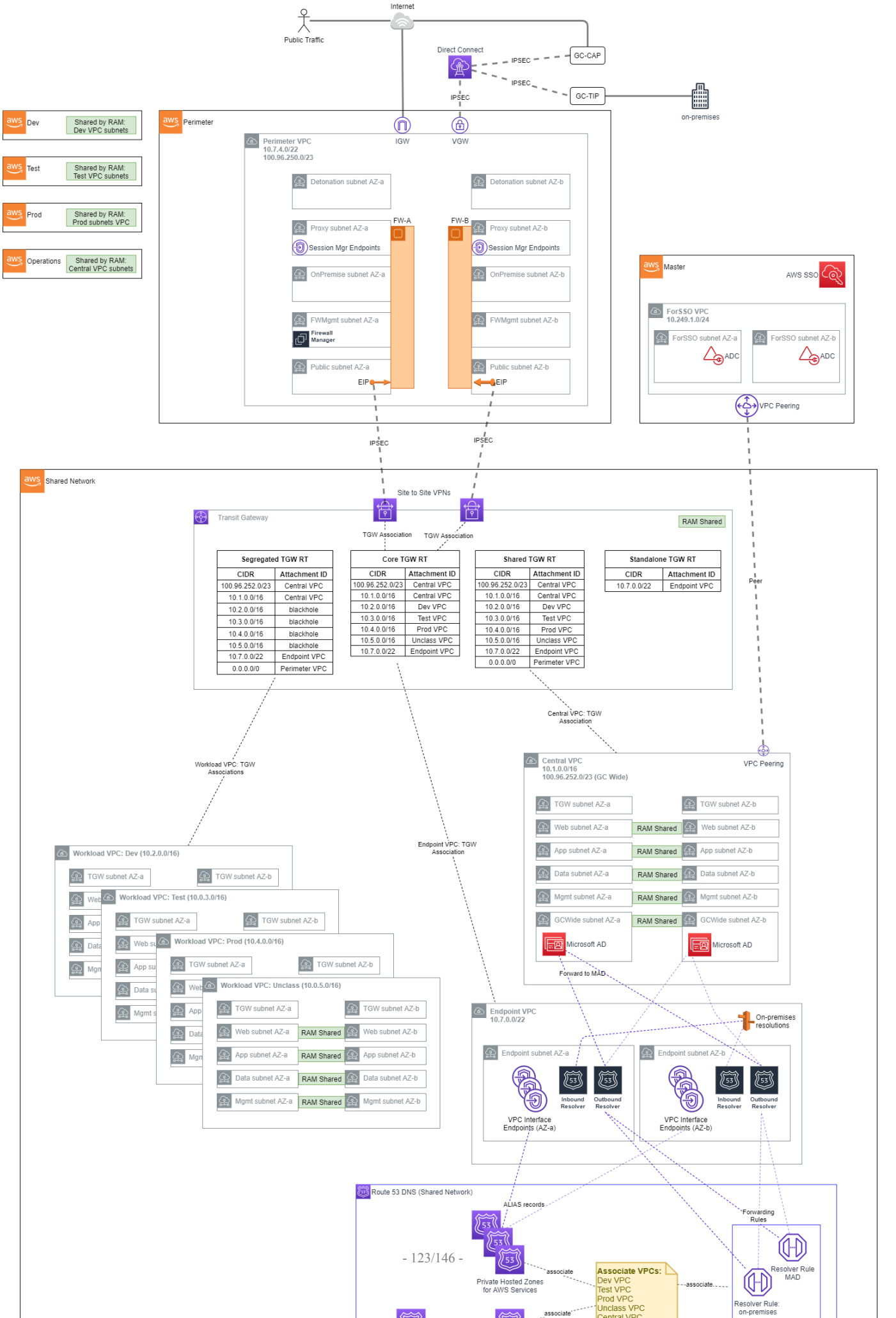
The *AWS Secure Environment Architecture* recommends that the AWS Private Marketplace is enabled for the Organization. Private Marketplace helps administrators govern which products they want their users to run on AWS by making it possible to see only products that comply with their organization's procurement policy. When Private Marketplace is enabled, it will replace the standard AWS Marketplace for all users.



7.1.3 Networking

Overview

The *AWS Secure Environment Architecture* networking is built on a principle of centralized on-premises and Internet ingress/egress, while enforcing data plane isolation between workloads in different environments. Connectivity to on-prem environments, internet egress, shared resources and AWS APIs are mediated at a central point of ingress/egress via the use of a Transit Gateway. Consider the following overall network diagram:


Internet
Public Traffic
Direct Connect
IPSEC
GC-CAP
GC-TIP
on-premises
Dev
Shared by RAM: Dev VPC subnets
Test
Shared by RAM: Test VPC subnets
Prod
Shared by RAM: Prod subnets VPC
Operations
Shared by RAM: Central VPC subnets
Perimeter
IGW
VGW
Perimeter VPC
10.7.4.0/22
100.96.250.0/23
Detonation subnet AZ-a
Detonation subnet AZ-b
Proxy subnet AZ-a
Session Mgr Endpoints
FW-A
FW-B
Proxy subnet AZ-b
OnPremise subnet AZ-a
OnPremise subnet AZ-b
FWMgmt subnet AZ-a
Firewall Manager
FWMgmt subnet AZ-b
Public subnet AZ-a
Public subnet AZ-b
EIP
Master
AWS SSO
ForSSO VPC
10.249.1.0/24
ForSSO subnet AZ-a
ForSSO subnet AZ-b
ADC
VPC Peering
Shared Network
Site to Site VPNs
Transit Gateway
RAM Shared
TGW Association
Segregated TGW RT
CIDR
Attachment ID
100.96.252.0/23
Central VPC
10.1.0.0/16
10.2.0.0/16
blackhole
10.3.0.0/16
10.4.0.0/16
10.5.0.0/16
10.7.0.0/22
Endpoint VPC
0.0.0.0/0
Perimeter VPC
Core TGW RT
Dev VPC
Test VPC
Prod VPC
Unclass VPC
Shared TGW RT
Standalone TGW RT
Peer
Central VPC: TGW Association
Workload VPC: TGW Associations
Endpoint VPC: TGW Association
Central VPC
10.1.0.0/16
100.96.252.0/23 (GC Wide)
TGW subnet AZ-a
TGW subnet AZ-b
Web subnet AZ-a
Web subnet AZ-b
App subnet AZ-a
App subnet AZ-b
Data subnet AZ-a
Data subnet AZ-b
Mgmt subnet AZ-a
Mgmt subnet AZ-b
GCWide subnet AZ-a
GCWide subnet AZ-b
Microsoft AD
Forward to MAD
Workload VPC: Dev (10.2.0.0/16)
Workload VPC: Test (10.0.3.0/16)
Workload VPC: Prod (10.4.0.0/16)
Workload VPC: Unclass (10.0.5.0/16)
Endpoint VPC
10.7.0.0/22
On-premises resolutions
Endpoint subnet AZ-a
Endpoint subnet AZ-b
Inbound Resolver
Outbound Resolver
VPC Interface Endpoints (AZ-a)
VPC Interface Endpoints (AZ-b)
Route 53 DNS (Shared Network)
ALIAS records
Forwarding Rules
Private Hosted Zones for AWS Services
associate
Associate VPCs:
Dev VPC
Test VPC
Prod VPC
Unclass VPC
Central VPC
Resolver Rule MAD
Resolver Rule: on-premises

All functional accounts use RAM-shared networking infrastructure as depicted above. The workload VPCs (Dev, Test, Prod, etc) are hosted in the Shared Network account and made available to the appropriate OU in the Organization.

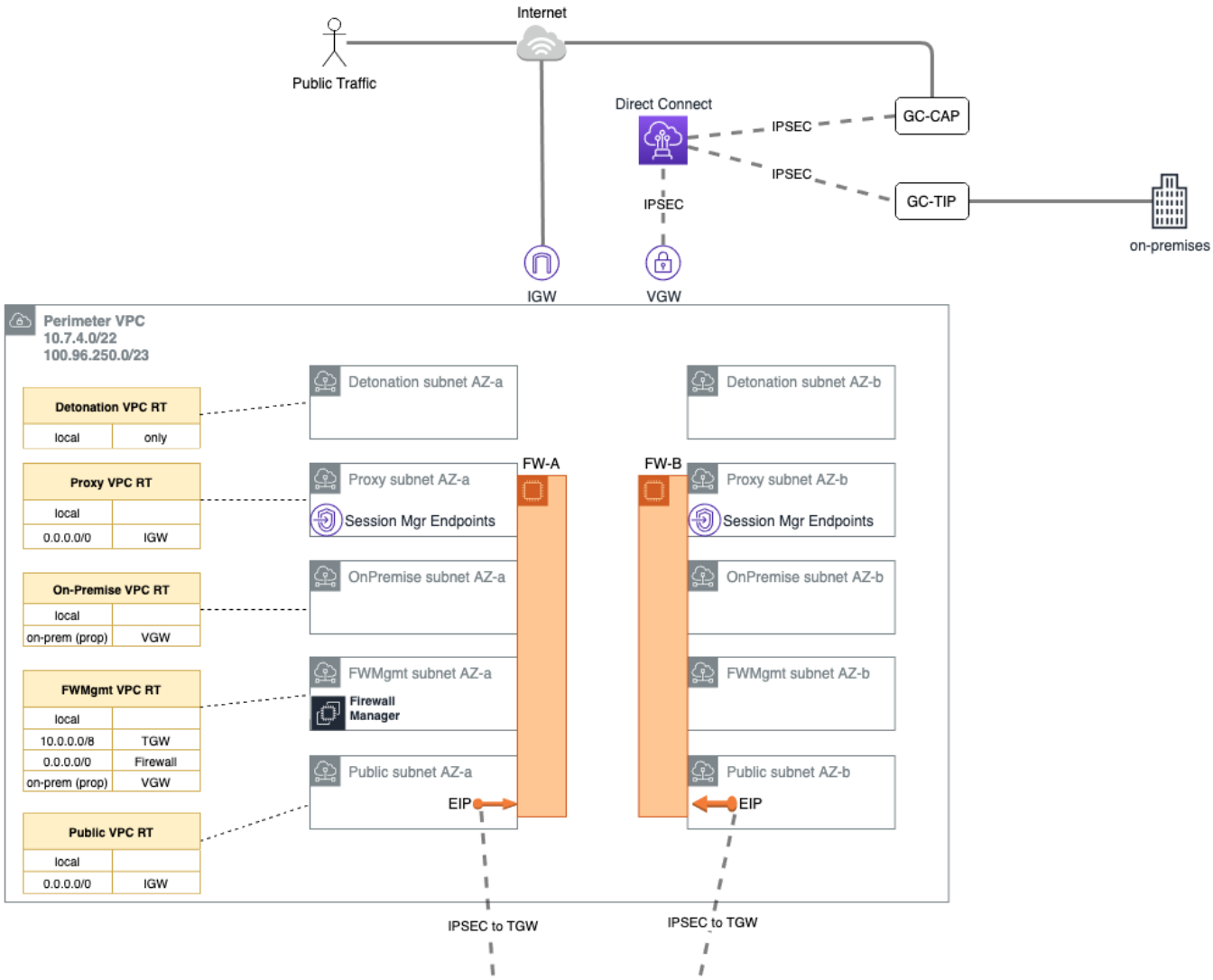
Perimeter

The perimeter VPC hosts the Organization's perimeter security services. The Perimeter VPC is used to control the flow of traffic between AWS Accounts and external networks for IaaS workloads: both public (internet) and in some cases private (access to on-premises datacenters). This VPC hosts Next Generation Firewalls (NGFW) that provide perimeter security services including virus scanning / malware protection, Intrusion Protection services, TLS Inspection and Web Application Firewall protection. If applicable, this VPC also hosts reverse proxy servers.

Note that this VPC is in its own isolated account, separate from Shared Network, in order to facilitate networking and security 'separation of duties'. Internal networking teams may administer the cloud networks in Shared Network without being granted permission to administer the security perimeter itself.

IP RANGES

- **Primary Range**: The *AWS Secure Environment Architecture* recommends that the perimeter VPC have a primary range in the RFC1918 block (e.g. `10.7.4.0/22`), used only for subnets dedicated to 'detonation' purposes. This primary range, in an otherwise-unused RFC1918 range, is not intended to be routable outside of the VPC, and is reserved for future use with malware detonation capabilities of NGFW devices.
- **Secondary Range**: This VPC should also have a secondary range in the RFC6598 block (e.g. `100.96.250.0/23`) used for the overlay network (NGFW devices inside VPN tunnel) for all other subnets. This secondary range is assigned by an external entity (e.g. Shared Services Canada), and should be carefully selected in order to co-exist with *AWS Secure Environment Architecture* deployments that exist at peer organizations; for instance other government departments that maintain a relationship with the same shared entity in a carrier-grade NAT topology. Although this is a 'secondary' range in VPC parlance, this VPC CIDR should be interpreted as the more 'significant' of the two with respect to Transit Gateway routing; the Transit Gateway will only ever interact with this 'secondary' range.



This VPC has four subnets per AZ, each of which hosts a port used by the NGFW devices, which are deployed in an HA pair. The purpose of these subnets is as follows.

- **Detonation**: This is an unused subnet reserved for future use with malware detonation capabilities of the NGFW devices.
- e.g. `10.7.4.0/24` - not routable except local.
- **Proxy**: This subnet hosts reverse proxy services for web and other protocols. It also contains the [three interface endpoints] necessary for AWS Systems Manager Session Manager, which enables SSH-less CLI access to authorized and authenticated principals in the perimeter account.
- e.g. `100.96.251.64/26`
- **On-Premises**: This subnet hosts the private interfaces of the firewalls, corresponding to connections from the on-premises network.
- e.g. `100.96.250.192/26`
- **FW-Management**: This subnet is used to host management tools and the management of the Firewalls itself.
- e.g. `100.96.251.160/27` - a smaller subnet is permissible due to modest IP requirements for management instances.
- **Public**: This subnet is the public-access zone for the perimeter VPC. It hosts the public interface of the firewalls, as well as application load balancers that are used to balance traffic across the firewall pair. There is one Elastic IPv4 address per public subnet that corresponds to the IPSec Customer Gateway (CGW) for the VPN connections into the Transit Gateway in Shared Networking.
- e.g. `100.96.250.0/26`

Outbound internet connections (for software updates, etc.) can be initiated from within the workload VPCs, and use the transparent proxy feature of the next-gen Firewalls.

Note on VPN Tunnel Redundancy: Each NGFW device manifests as a unique CGW on the AWS side (shared network account) of the IPSec VPNs. Moreover, there are **two** *Site-to-Site VPNs* in this architecture, each with one active tunnel (and one inactive tunnel); taken together, the pair is redundant. In many hybrid networking configurations, a single *Site-to-Site VPN* resource is used with dual active tunnels. Customers may receive the following email notification from the AWS VPC service team:

> You're receiving this message because you have at least one VPN Connection in the ca-central-1 Region, for which your VPN Customer Gateway is not using both tunnels. This mode of operation is not recommended as you may experience connectivity issues if your active tunnel fails.

This message may be disregarded, as it is premised on a traditional hybrid configuration with dual tunnels. Customers may create a VPN support case (in shared network account) requesting that these informational emails are disabled.

Shared Network

The shared network account, and the AWS networking resources therein, form the core of the cloud networking infrastructure across the account structure. Rather than the individual accounts defining their own networks, these are instead centralized here and shared out to the relevant OUs. Principals in a Dev OU will have access to a Dev VPC, Test OU will have access to a Test VPC and so on - all of which are owned by this account.

You can share AWS Transit Gateways, Subnets, AWS License Manager configurations, and Amazon Route 53 Resolver rules resources with AWS Resource Access Manager (RAM). The RAM service eliminates the need to create duplicate resources in multiple accounts, reducing the operational overhead of managing those resources in every single account.

TRANSIT GATEWAY

The Transit Gateway is a central hub that performs several core functions within the Shared Network account.

1. Routing of permitted flows; for example a Workload to On-premises via the Perimeter VPC.
2. All routing tables in SharedNetwork VPCs send `0.0.0.0/0` traffic to the TGW, where its handling will be determined by the TGW Route Table (TGW-RT) that its attachment is associated with. For example:

- an HTTP request to `registry.hub.docker.com` from the Test VPC will go to the TGW
- The Segregated TGW RT will direct that traffic to the Perimeter VPC via the IPsec VPNs
- The request will be proxied to the internet, via GC-CAP if appropriate
- The return traffic will again transit the IPsec VPNs
- The `10.3.0.0/16` bound response traffic will arrive at the Core TGW RT, where a propagation in that TGW RT will direct the response back to the Test VPC.

3. Defining separate routing domains that prohibit undesired east-west flows at the network level; for example, by prohibiting Dev to Prod traffic. For example:
4. All routing tables in SharedNetwork VPCs send `0.0.0.0/0` traffic to the TGW, which defines where the next permissible hop is. For example, `10.2.0.0/16` Dev traffic destined for the `10.0.4.0/16` Prod VPC will be blocked by the blackhole route in the Segregated TGW RT.
5. Enabling centralization of shared resources; namely a shared Microsoft AD installation in the Central VPC, and access to shared VPC Endpoints in the Endpoint VPC.
6. The Central VPC, and the Endpoint VPC are routable from Workload VPCs. This provides an economical way to share Organization wide resources that are nonetheless isolated into their own VPCs. For example:

- a `git` request in the `Dev` VPC to `git.private-domain.ca` resolves to a `10.1.0.0/16` address in the `Central` VPC.
- The request from the `Dev` VPC will go to the TGW due to the VPC routing table associated with that subnet
- The TGW will send the request to the `Central` VPC via an entry in the Segregated TGW RT
- The `git` response will go to the TGW due to the VPC routing table associated with that subnet
- The Shared TGW RT will direct the response back to the `Dev` VPC

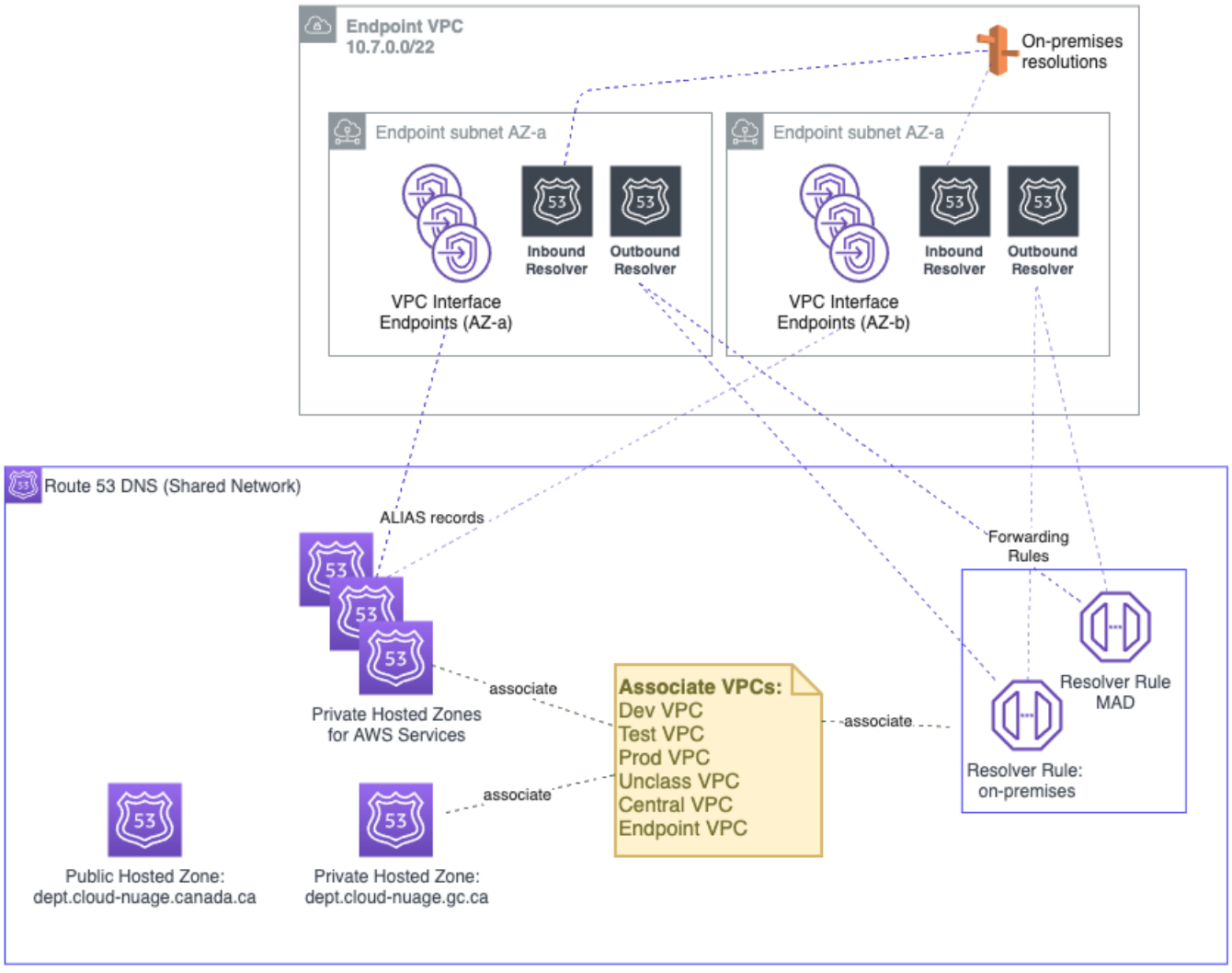
The four TGW RTs exist to serve the following main functions:

- **Segregated TGW RT**: Used as the association table for the workload VPCs; prevents east-west traffic, except to shared resources.
- **Core TGW RT**: Used for internet/on-premises response traffic, and Endpoint VPC egress.
- **Shared TGW RT**: Used to provide `Central` VPC access east-west for the purposes of response traffic to shared workloads
- **Standalone TGW RT**: Reserved for future use. Prevents TGW routing except to the Endpoint VPC.

Note that a unique BGP ASN will need to be available for the TGW.

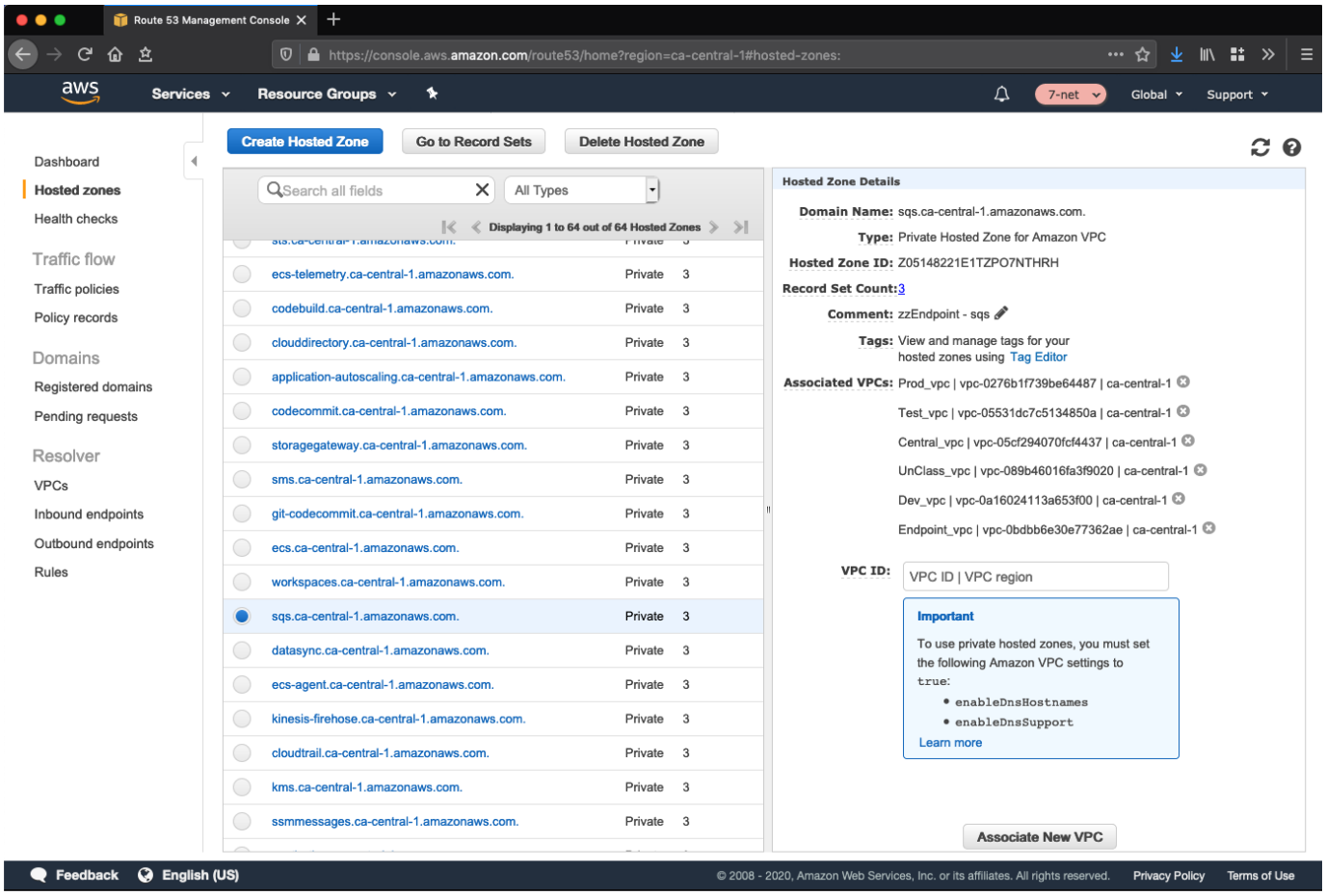
ENDPOINT VPC

DNS functionality for the network architecture is centralized in the Endpoint VPC. It is recommended that the Endpoint VPC use a RFC1918 range - e.g. `10.7.0.0/22` with sufficient capacity to support 60+ AWS services and future endpoint expansion, and inbound and outbound resolvers (all figures per AZ).



ENDPOINT VPC: INTERFACE ENDPOINTS

The endpoint VPC hosts VPC Interface Endpoints (VPCEs) and associated Route 53 private hosted zones for all applicable services in the `ca-central-1` region. This permits traffic destined for an eligible AWS service; for example SQS, to remain entirely within the SharedNetwork account rather than transiting via the IPv4 public endpoint for the service:



From within an associated workload VPC such as `Dev`, the service endpoint (e.g. `sqs.ca-central-1.amazonaws.com`) will resolve to an IP in the `Endpoint` VPC:

```
sh-4.2$ nslookup sqs.ca-central-1.amazonaws.com
Server:         10.2.0.2                # Dev VPC's .2 resolver.
Address:        10.2.0.2#53

Non-authoritative answer:
Name:   sqs.ca-central-1.amazonaws.com
Address: 10.7.1.190                     # IP in Endpoint VPC - AZ-a.
Name:   sqs.ca-central-1.amazonaws.com
Address: 10.7.0.135                     # IP in Endpoint VPC - AZ-b.
```

This cross-VPC resolution of the service-specific private hosted zone functions via the association of each VPC to each private hosted zone, as depicted above.

ENDPOINT VPC: HYBRID DNS

The Endpoint VPC also hosts the common DNS infrastructure used to resolve DNS queries:

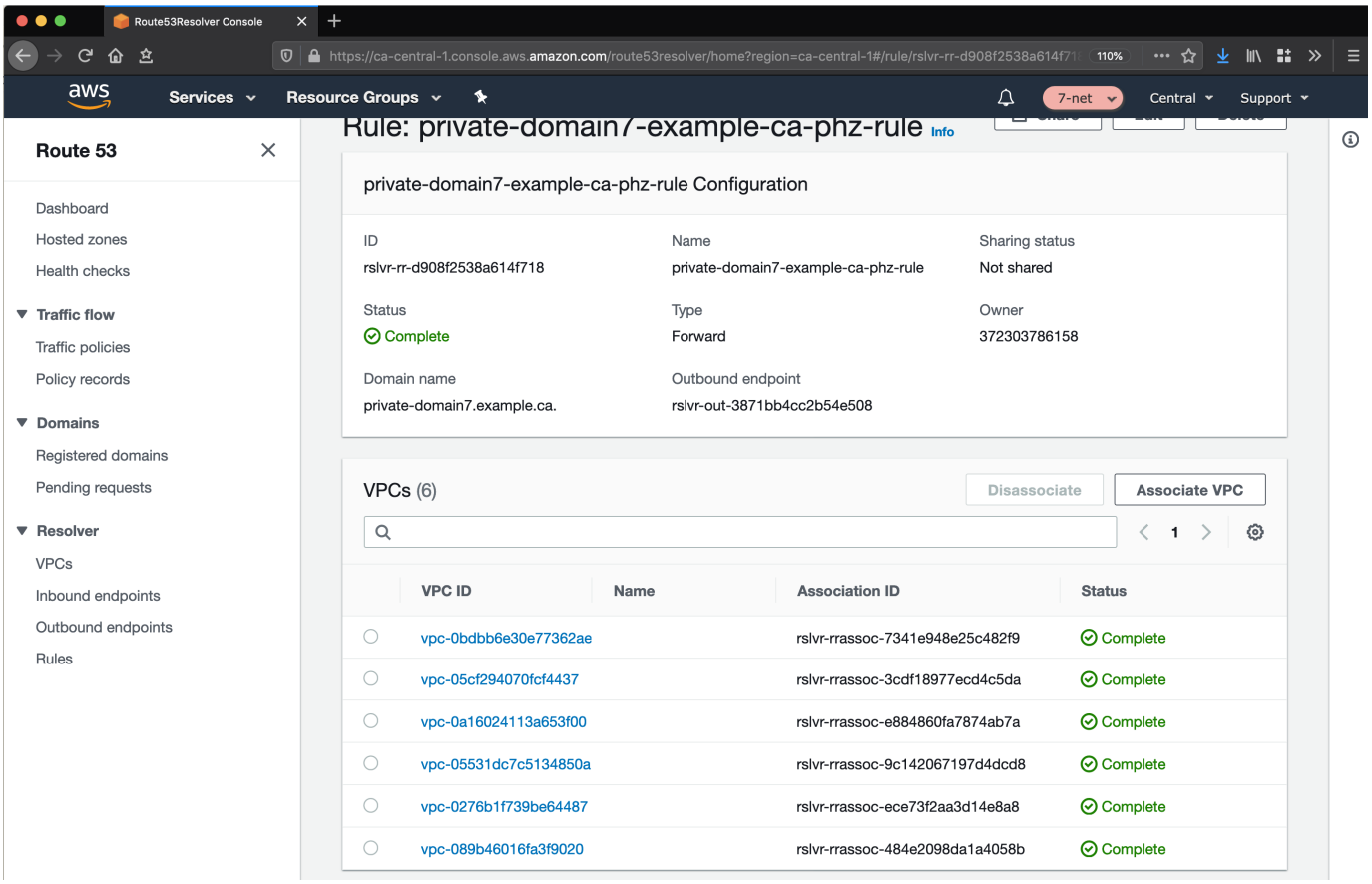
- within the cloud
- from the cloud to on-premises
- from on-premises to the cloud

Within The Cloud

In-cloud DNS resolution applies beyond the DNS infrastructure that is put in place to support the Interface Endpoints for the AWS services in-region. Other DNS zones, associated with the Endpoint VPC, are resolvable the same way via an association to workload VPCs.

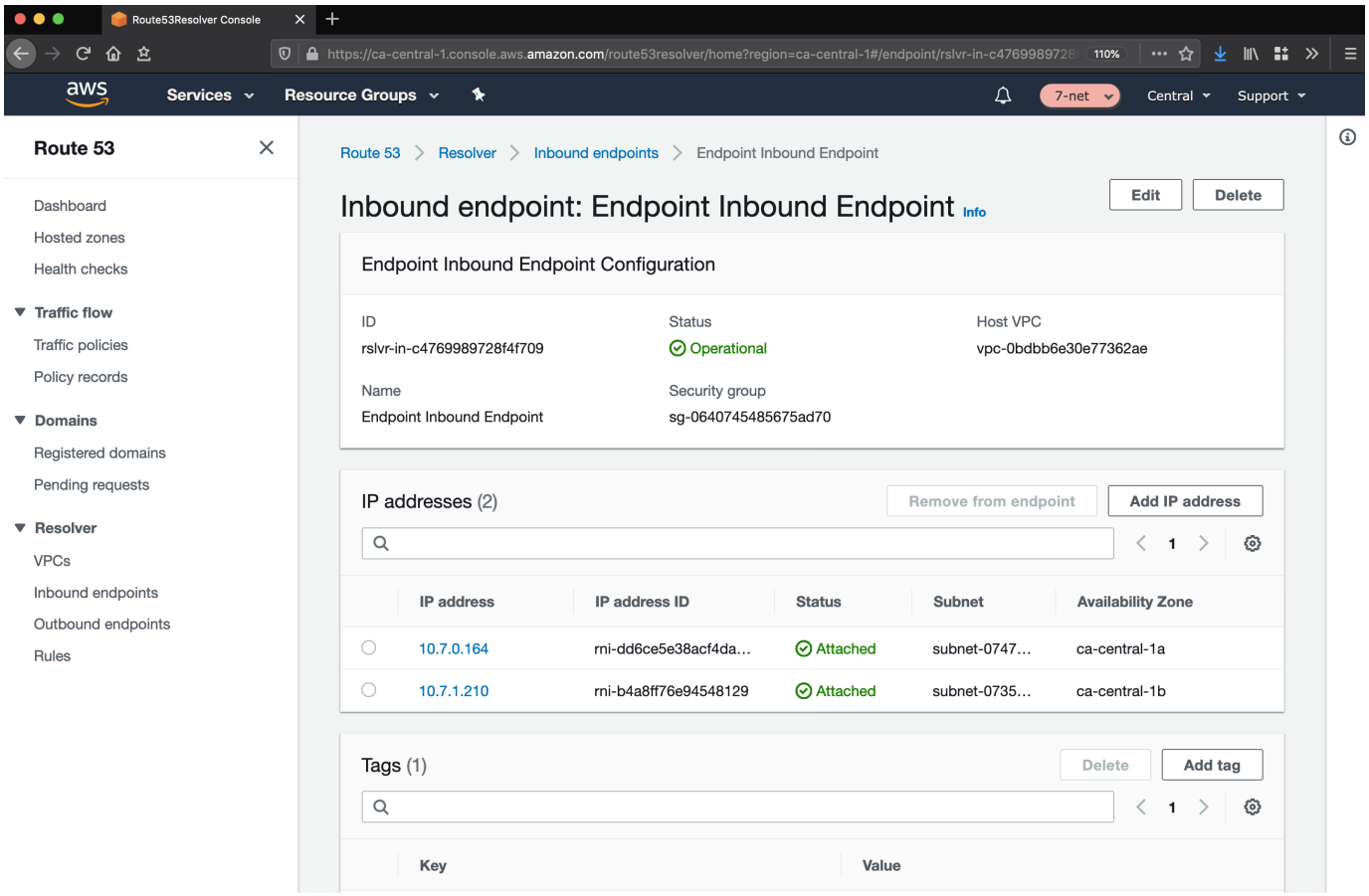
From Cloud to On-Premises

DNS Resolution from the cloud to on-premises is handled via the use of a Route 53 Outbound Endpoint, deployed in the Endpoint VPC, with an associated Resolver rule that forwards DNS traffic to the outbound endpoint. Each VPC is associated to this rule.



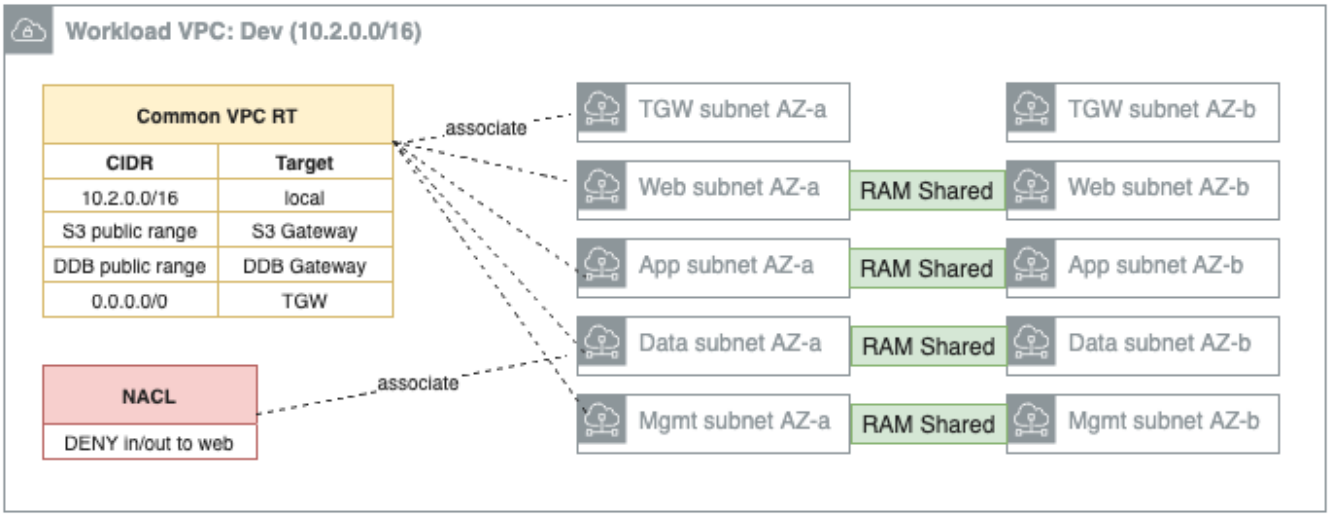
From On-Premises to Cloud

Conditional forwarding from on-premises networks is made possible via the use of a Route 53 Inbound Endpoint. On-prem networks send resolution requests for relevant domains to the endpoints deployed in the Endpoint VPC:



WORKLOAD VPCS

The workload VPCs are where line of business applications ultimately reside, segmented by environment (`Dev` , `Test` , `Prod` , etc). It is recommended that the Workload VPC use a RFC1918 range (e.g. `10.2.0.0/16` for `Dev` , `10.3.0.0/16` for `Test` , etc).



Note that security groups are recommended as the primary data-plane isolation mechanism between applications that may coexist in the same VPC. It is anticipated that unrelated applications would coexist in their respective tiers without ever permitting east-west traffic flows.

The following subnets are defined by the *AWS Secure Environment Architecture*:

- **TGW subnet**: This subnet hosts the elastic-network interfaces for the TGW attachment. A `/27` subnet is sufficient.
- **Web subnet**: This subnet hosts front-end or otherwise 'client' facing infrastructure. A `/20` or larger subnet is recommended to facilitate auto-scaling.
- **App subnet**: This subnet hosts app-tier code (EC2, containers, etc). A `/19` or larger subnet is recommended to facilitate auto-scaling.
- **Data subnet**: This subnet hosts data-tier code (RDS instances, ElastiCache instances). A `/21` or larger subnet is recommended.
- **Mgmt subnet**: This subnet hosts bastion or other management instances. A `/21` or larger subnet is recommended.

Each subnet is associated with a Common VPC Route Table, as depicted above. Gateway Endpoints for relevant services (Amazon S3, Amazon DynamoDB) are installed in the Common route tables of all Workload VPCs. Aside from local traffic or gateway-bound traffic, `0.0.0.0/0` is always destined for the TGW.

Security Groups

Security Groups are instance level firewalls, and represent a foundational unit of network segmentation across AWS networking. Security groups are stateful, and support ingress/egress rules based on protocols and source/destinations. While CIDR ranges are supported by the latter, it is preferable to instead use other security groups as source/destinations. This permits a higher level of expressiveness that is not coupled to particular CIDR choices and works well with autoscaling; e.g.

"permit port 3306 traffic from the `App` tier to the `Data` tier"

versus

"permit port 3306 traffic from `10.0.1.0/24` to `10.0.2.0/24`.

Security group egress rules are often used in 'allow all' mode (`0.0.0.0/0`), with the focus primarily being on consistently whitelisting required ingress traffic. This ensures day to day activities like patching, access to DNS, or to directory services access can function on instances without friction. The provided sample security groups in the workload accounts offers a good balance that considers both security, ease of operations, and frictionless development. They allow developers to focus on developing, enabling them to simply use the pre-created security constructs for their workloads, and avoid the creation of wide-open security groups. Developers can equally choose to create more appropriate least-privilege security groups more suitable for their application, if they are skilled in this area. It is expected as an application is promoted through the SDLC cycle from Dev through Test to Prod, these security groups will be further refined by the extended customers teams to further reduce privilege, as appropriate. It is expected that each customer will review and tailor their Security Groups based on their own security requirements.

NACLs

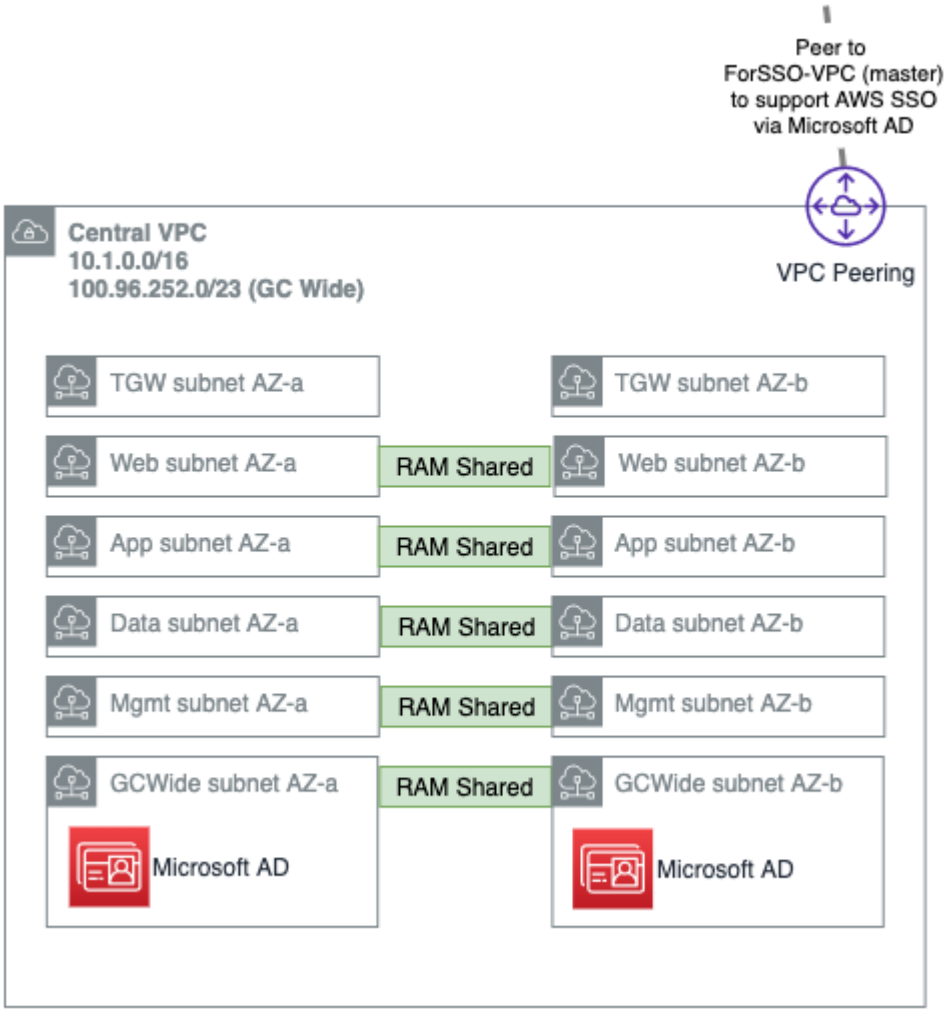
Network Access-Control Lists (NACLs) are stateless constructs used sparingly as a defense-in-depth measure in this architecture. AWS generally discourages the use of NACLs given the added complexity and management burden, given the availability and ease of use provided by security groups. Each network flow often requires four NACL entries (egress from ephemeral, ingress to destination, egress from destination, ingress to ephemeral). The architecture recommends NACLs as a segmentation mechanism for `Data` subnets; i.e. `DENY` all inbound traffic to such a subnet except that which originates in the `App` subnet for the same VPC. As with security groups, we encourage customers to review and tailor their NACLs based on their own security requirements.

CENTRAL VPC

The Central VPC is a network for localizing operational infrastructure that may be needed across the Organization, such as code repositories, artifact repositories, and notably, the managed Directory Service (Microsoft AD). Instances that are domain joined will connect to this AD domain - a network flow that is made possible from anywhere in the network structure due to the inclusion of the Central VPC in all relevant association TGW RTs.

It is recommended that the Central VPC use a RFC1918 range (e.g. `10.1.0.0/16`) for the purposes of routing from the workload VPCs, and a secondary range from the RFC6598 block (e.g. `100.96.252.0/23`) to support the Microsoft AD workload.

Note that this VPC also contains a peering relationship to the `ForSSO` VPC in the Organization Management (root) account. This exists purely to support connectivity from an AD-Connector instance in the Organization Management (root) account, which in turn enables AWS SSO for federated login to the AWS control plane.



Domain Joining

An EC2 instance deployed in the Workload VPCs can join the domain corresponding to the Microsoft AD in `Central` provided the following conditions are all true:

1. The instance needs a network path to the Central VPC (given by the Segregated TGW RT), and appropriate security group assignment
2. The Microsoft AD should be 'shared' with the account the EC2 instance resides in (The *AWS Secure Environment Architecture* recommends these directories are shared to workload accounts)
3. The instance has the AWS managed policies `AmazonSSMManagedInstanceCore` and `AmazonSSMDirectoryServiceAccess` attached to its IAM role, or runs under a role with at least the permission policies given by the combination of these two managed policies.
4. The EC2's VPC has an associated resolver rule that directs DNS queries for the AD domain to the Central VPC.

SANDBOX VPC

A sandbox VPC, not depicted, may be included in the *AWS Secure Environment Architecture*. This is **not** connected to the Transit Gateway, Perimeter VPC, on-premises network, or other common infrastructure. It contains its own Internet Gateway, and is an entirely separate VPC with respect to the rest of the *AWS Secure Environment Architecture*.

The sandbox VPC should be used exclusively for time-limited experimentation, particularly with out-of-region services, and never used for any line of business workload or data.

7.1.4 Authorization and Authentication

The *AWS Secure Environment Architecture* makes extensive use of AWS authorization and authentication primitives from the Identity and Access Management (IAM) service as a means to enforce the guardrail objectives of the *AWS Secure Environment Architecture*, and govern access to the set of accounts that makes up the Organization.

Relationship to the Organization Management (root) AWS Account

AWS accounts, as a default position, are entirely self-contained with respect to IAM principals - their Users, Roles, Groups are independent and scoped only to themselves. Accounts created by AWS Organizations deploy a default role with a trust policy back to the Organization Management (root). By default, this role is named the `OrganizationAccountAccessRole`; by contrast, the *AWS Secure Environment Architecture* allows customers to customize this role by defining it in `organization-admin-role` (default: `AWSCloudFormationStackSetAdministrationRole`).

```
{
  "Role": {
    "Path": "/",
    "RoleName": "AWSCloudFormationStackSetExecutionRole",
    "Arn": "arn:aws:iam::111111111111:role/AWSCloudFormationStackSetExecutionRole", // Child account.
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "AWS": "arn:aws:iam::123456789012:root" // Organization Management (root) account may assume this role.
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
}
```

As discussed, the AWS Organization resides in the Organization Management (root) account. This account is not used for workloads and is primarily a gateway to the entire cloud footprint for a high-trust principal. This is realized via the role defined in `organization-admin-role` (default: `AWSCloudFormationStackSetAdministrationRole`). It is therefore crucial that the Organization Management (root) account root credentials be handled with extreme diligence, and with a U2F hardware key enabled as a second-factor (and stored in a secure location such as a safe).

Break Glass Accounts

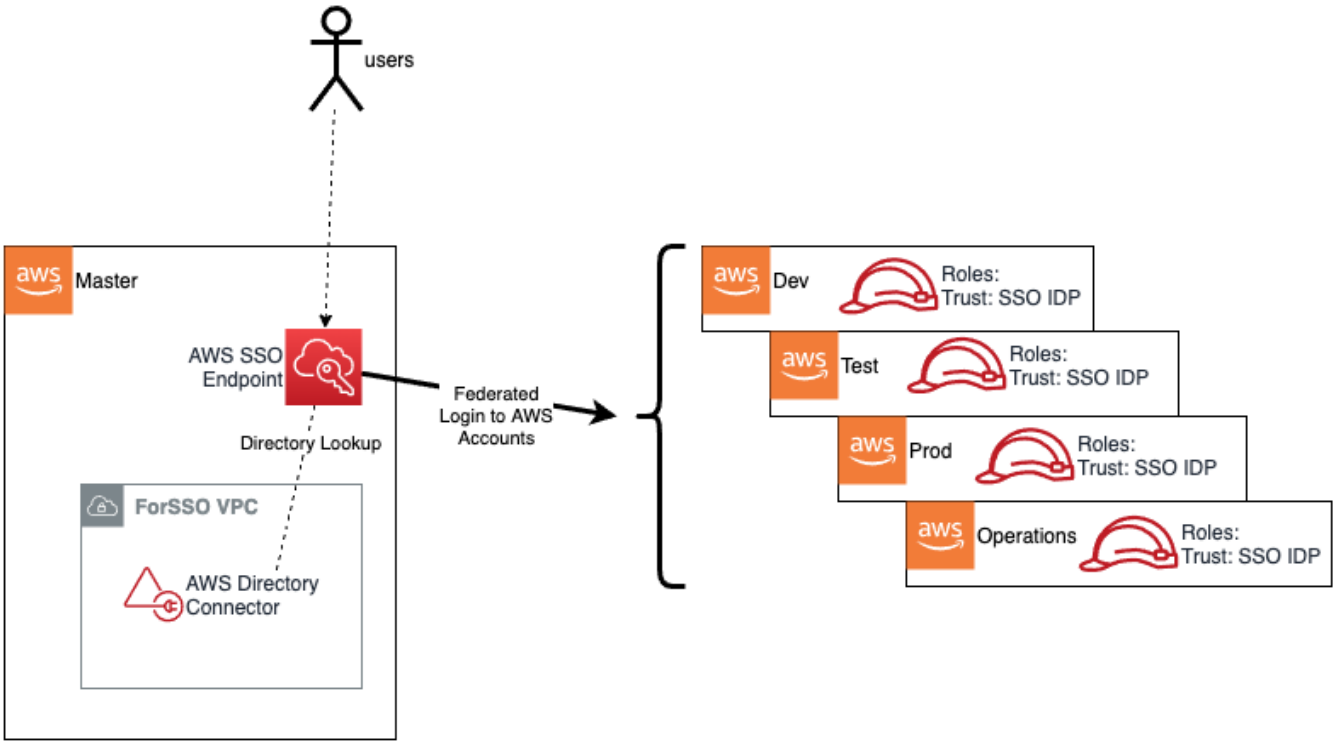
Given the Organizational-wide trust relationship to the role defined in `organization-admin-role` (default: `AWSCloudFormationStackSetAdministrationRole`) and its broad exclusion from SCPs (discussed below), the assumption of this role grants 'super admin' status, and is thus an extremely high privilege operation. The ability to assume this role should be considered a 'break glass' capability - to be used only in extraordinary circumstances. Access to this role can be granted by IAM Users or IAM Roles in the Organization Management (root) account (via SSO) - as with the Organization Management (root) account credentials, these should be handled with extreme diligence, and with a U2F hardware key enabled as a second-factor (and stored in a secure location such as a safe).

TBD: This role was locked down starting in v1.2.5 - Add further details here /TODO

Control Plane Access via AWS SSO

The vast majority of end-users of the AWS cloud within the Organization will never use or interact with the Organization Management (root) account, or indeed the root users of any child account in the Organization. The *AWS Secure Environment Architecture* recommends instead that AWS SSO be provisioned in the Organization Management (root) account (a rare case where Organization Management (root) account deployment is mandated).

Users will login to AWS via the web-based endpoint for the AWS SSO service:



Via an AWS Directory Connector deployed in the Organization Management (root) account, AWS SSO will authenticate the user based on the underlying Microsoft AD installation (in the Central account). Based on group membership, the user will be presented with a set of roles to assume into those accounts. For example, a developer may be placed into groups that permit `Admin` access in the `Dev` account and `Readonly` access in `Test`; meanwhile an IT Director may have high-privilege access to most, or all, accounts. In effect, AWS SSO adds SAML IdP capabilities to the AWS Managed Microsoft AD, with the AWS Console acting as a service-provider (SP) in SAML parlance. Other SAML-aware SPs may also be used with AWS SSO.

SSO USER ROLES

AWS SSO creates an identity provider (IdP) in each account in the Organization. The roles used by end users have a trust policy to this IdP. When a user authenticates to AWS SSO (via the underlying AD Connector) and selects a role to assume based on their group membership, the SSO service provides the user with temporary security credentials unique to the role session. In such a scenario, the user has no long-term credentials (e.g. password, or access keys) and instead uses their temporary security credentials.

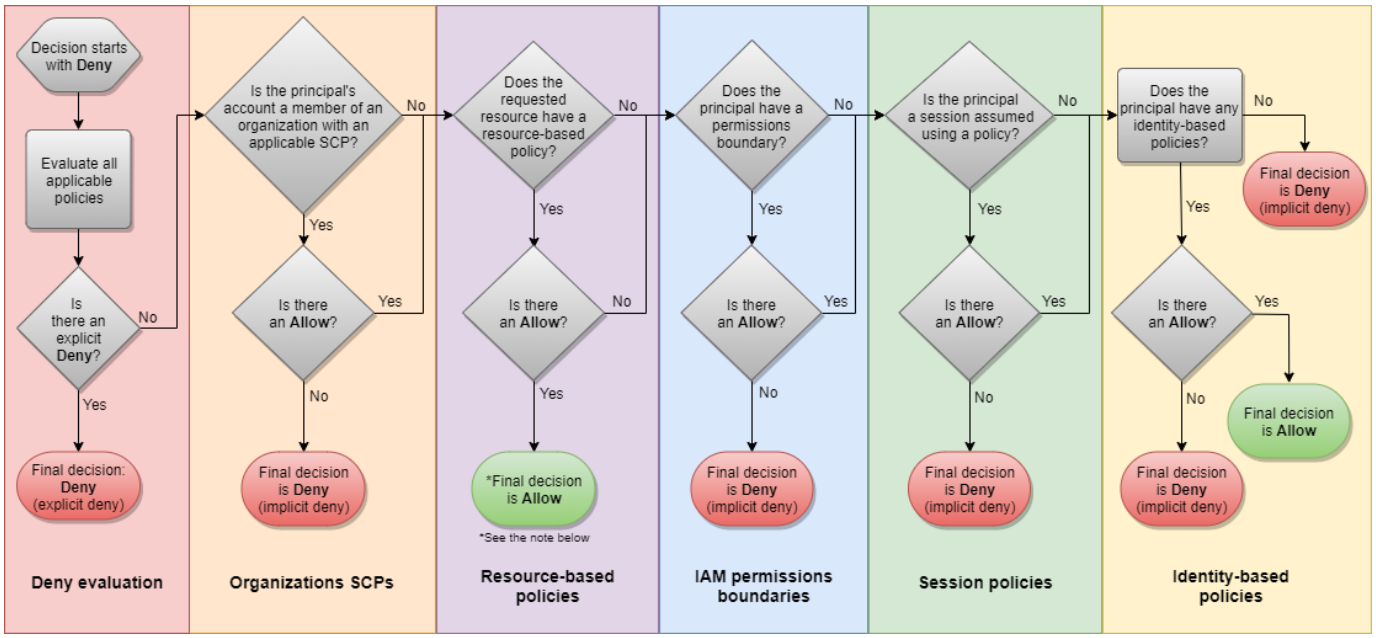
Users, via their AD group membership, are ultimately assigned to SSO User Roles via the use of AWS SSO Permission Sets. A permission set is an assignment of a particular permission policy to a set of accounts. For example:

An organization might decide to use **AWS Managed Policies for Job Functions** that are located within the SSO service as the baseline for role-based-access-control (RBAC) separation within an AWS account. This enables job function policies such as:

- **Administrator** - This policy grants almost all actions for all AWS services and for all resources in the account.
- **Developer Power User** - This user performs application development tasks and can create and configure resources and services that support AWS aware application development.
- **Database Administrator** - This policy grants permissions to create, configure, and maintain databases. It includes access to AWS database services, such as Amazon DynamoDB, Amazon Relational Database Service (RDS), and Amazon Redshift.
- **View-Only User** - This policy grants `List*`, `Describe*`, `Get*`, `View*`, and `Lookup*` access to resources for most AWS services.

PRINCIPAL AUTHORIZATION

Having assumed a role, a user's permission-level within an AWS account with respect to any API operation is governed by the IAM policy evaluation logic flow (detailed here):



Having an `Allow` to a particular API operation from the Role (i.e. Session Policy) does not necessarily imply that API operation will succeed. As depicted above, **Deny** may result due to another evaluation stage in the logic; for example a restrictive permission boundary or an explicit `Deny` at the Resource or SCP (account) level. SCPs are used extensively as a guardrailing mechanism in the *AWS Secure Environment Architecture*, and are discussed in a later section.

Root Authorization

Root credentials for individual accounts in an AWS organization may be created on demand via a password reset process on the unique account email address; however, the *AWS Secure Environment Architecture* specifically denies this via SCP. Root credentials authorize all actions for all AWS services and for all resources in the account (except anything denied by SCPs). There are some actions which only root has the capability to perform which are found within the AWS online documentation. These are typically rare operations (e.g. creation of X.509 keys), and should not be required in the normal course of business. Any root credentials, if ever they need to be created, should be handled with extreme diligence, with U2F MFA enabled.

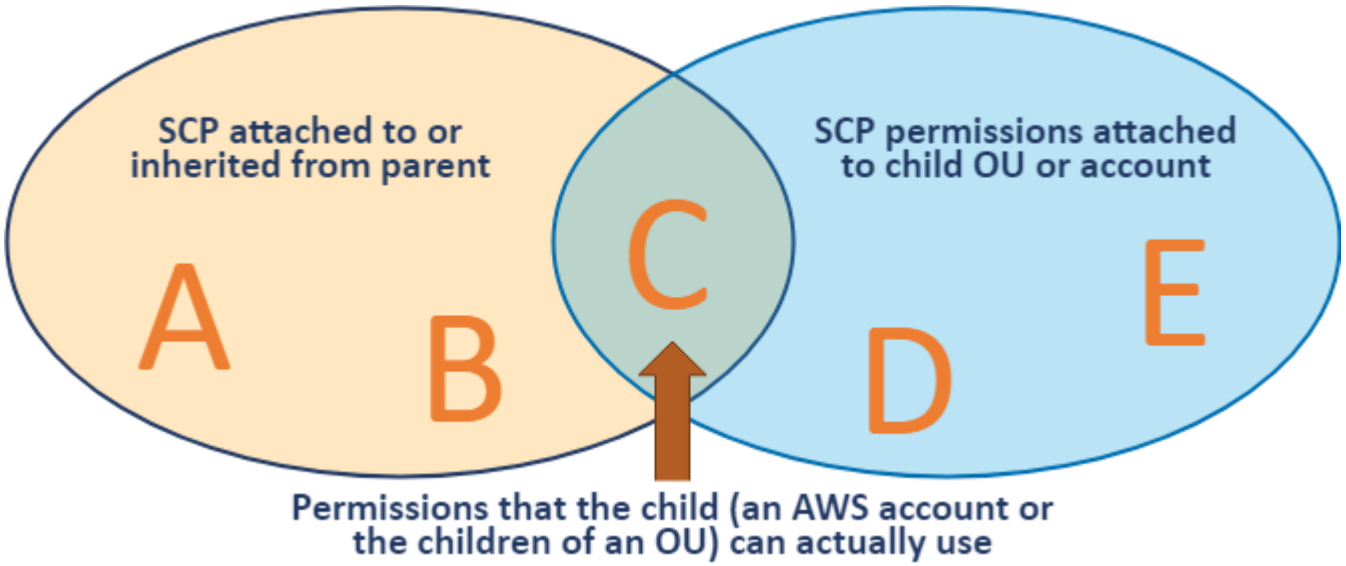
Service Roles

A service role is an IAM Role that a service assumes to perform actions in an account on the user's behalf. When a user sets up AWS service environments, the user must define an IAM Role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles provide access only within a single account and cannot be used to grant access to services in other accounts. Users can create, modify, and delete a service role from within the IAM service. For example, a user can create a role that allows Amazon Redshift to access an Amazon S3 bucket on the user's behalf and then load data from that bucket into an Amazon Redshift cluster. In the case of SSO, during the process in which AWS SSO is enabled, the AWS Organizations service grants AWS SSO the necessary permissions to create subsequent IAM Roles.

Service Control Policies

Service Control Policies are a key preventative control recommended by the *AWS Secure Environment Architecture*. It is crucial to note that SCPs, by themselves, never *grant* permissions. They are most often used to `Deny` certain actions at a root, OU, or account level within an AWS Organization. Since `Deny` always overrides `Allow` in the IAM policy evaluation logic, SCPs can have a powerful effect on all principals in an account, and can wholesale deny entire categories of actions irrespective of the permission policy attached to the principal itself - even the root user of the account.

SCPs follow an inheritance pattern from the root of the Organization:



In order for any principal to be able to perform an action A, it is necessary (but not sufficient) that there is an `Allow` on action A from all levels of the hierarchy down to the account, and no explicit `Deny` anywhere. This is discussed in further detail in How SCPs Work.

The *AWS Secure Environment Architecture* recommends the following SCPs in the Organization:

PBMM ONLY

This is a comprehensive policy whose main goal is to provide a PBMM-compliant cloud environment, namely prohibiting any non-centralized networking, and mandating data residency in Canada. It should be attached to all non-`Unclass` OUs.

Policy Statement ID (SID)	Description
`DenyNetworkPBMMONLY`	Prevents the creation of any networking infrastructure in the workload accounts such as VPCs, NATs, VPC peers, etc.
`DenyAllOutsideCanadaPBMMONLY`	Prevents the use of any service in any non-Canadian AWS region with the exception of services that are considered global; e.g. CloudFront, IAM, STS, etc
`ScopeSpecificGlobalActionsToCanadaUSE1`	Within services that are exempted from `DenyAllOutsideCanadaPBMMONLY`, scope the use of those services to the `us-east-1` region

PBMM UNCLASS ONLY

This is broadly similar to `PBMM Only`; however it relaxes the requirement for Canadian region usage, and does not prohibit network infrastructure creation (e.g. VPCs, IGWs). This is appropriate for OUs in which AWS service experimentation is taking place.

Policy Statement ID (SID)	Description
`DenyUnclass`	Prevents the deletion of KMS encryption keys and IAM password policies
`DenyAllOutsideCanadaUS`	Prevents the use of any service in any region that is not `ca-central-1` or `us-east-1`, with the exception of services that are considered global; e.g. CloudFront, IAM, STS, etc

PBMM GUARDRAILS (PARTS 1 AND 2)

PBMM Guardrails apply across the Organization. These guardrails protect key infrastructure, mandate encryption at rest, and prevent other non-PBMM configurations. Note that this guardrail is split into two parts due to a current limitation of SCP sizing, but logically it should be considered a single policy.

Policy Statement ID (SID)	Description
`DenyTag1`	Prevents modification of any protected security group
`DenyTag2`	Prevents modification of any protected IAM resource
`DenyS3`	Prevents modification of any S3 bucket used for Accelerator purposes
`ProtectCloudFormation`	Prevents modification of any CloudFormation stack used for Accelerator tool purposes
`DenyAlarmDeletion`	Prevents modification of any cloudwatch alarm used to alert on significant control plane events
`ProtectKeyRoles`	Prevents any IAM operation on Accelerator tool IAM roles
`DenySSMDel`	Prevents modification of any ssm resource used for Accelerator tool purposes
`DenyLogDel`	Prevents the deletion of any log resource in Cloudwatch Logs
`DenyLeaveOrg`	Prevents an account from leaving the Organization
`DenyLambdaDel`	Prevents the modification of any guardrail Lambda function
`BlockOther`	Prevents miscellaneous operations; e.g. Deny `ds:DisableSso`
`BlockMarketplacePMP`	Prevents the modification or creation of a cloud private marketplace
`DenyRoot`	Prevents the use of the root user in an account
`EnforceEbsEncryption`	Enforces the use of volume level encryption in running instances
`EnforceEBSVolumeEncryption`	Enforces the use of volume level encryption with EBS
`EnforceRdsEncryption`	Enforces the use of RDS encryption
`EnforceAuroraEncryption`	Enforces the use of Aurora encryption
`DenyRDGWRole`	Prevents the modification of a role used for Remote Desktop Gateway
`DenyGDSHFMAAChange`	Prevents the modification of GuardDuty & Security Hub

Encryption at Rest

Note that the `*Encryption*` SCP statements above, taken together, mandate encryption at rest for block storage volumes used in EC2 and RDS instances.

QUARANTINE DENY ALL

This policy can be attached to an account to 'quarantine' it - to prevent any AWS operation from taking place. This is useful in the case of an account with credentials which are believed to have been compromised.

Policy Statement ID (SID)	Description
`DenyAllAWSServicesExceptBreakglassRoles`	Blanket denial on all AWS control plane operations for all non-break-glass roles

QUARANTINE NEW OBJECT

This policy is applied to new accounts upon creation. After the installation of guardrails, it is removed. In the meantime, it prevents all AWS control plane operations except by principals required to deploy guardrails.

Policy Statement ID (SID)	Description
`DenyAllAWSServicesExceptBreakglassRoles`	Blanket denial on all AWS control plane operations for all non-break-glass roles

7.1.5 Logging and Monitoring

The *AWS Secure Environment Architecture* recommends the following detective controls across the Organization. These controls, taken together, provide a comprehensive picture of the full set of control plane and data plane operations across the set of accounts.

CloudTrail

A CloudTrail Organizational trail should be deployed into the Organization. For each account, this captures management events and optionally S3 data plane events taking place by every principal in the account. These records are sent to an S3 bucket in the log archive account, and the trail itself cannot be modified or deleted by any principal in any child account. This provides an audit trail for detective purposes in the event of the need for forensic analysis into account usage. The logs themselves provide an integrity guarantee: every hour, CloudTrail produces a digest of that hour's logs files, and signs with its own private key. The authenticity of the logs may be verified using the corresponding public key. This process is detailed here.

VPC Flow Logs

VPC Flow Logs capture information about the IP traffic going to and from network interfaces in an AWS Account VPC such as source and destination IPs, protocol, ports, and success/failure of the flow. The *AWS Secure Environment Architecture* recommends enabling `ALL` (i.e. both accepted and rejected traffic) logs for all VPCs in the Shared Network account with an S3 destination in the log-archive account. More details about VPC Flow Logs are available here.

Note that certain categories of network flows are not captured, including traffic to and from Traffic to and from `169.254.169.254` for instance metadata, and DNS traffic with an Amazon VPC resolver.

GuardDuty

Amazon GuardDuty is a threat detection service that continuously monitors for malicious activity and unauthorized behavior to protect your AWS accounts and workloads. The service uses machine learning, anomaly detection, and integrated threat intelligence to identify and prioritize potential threats. GuardDuty uses a number of data sources including VPC Flow Logs and CloudTrail logs.

The *AWS Secure Environment Architecture* recommends enabling GuardDuty at the Organization level, and delegating the security account as the GuardDuty Administrative account. The GuardDuty Administrative account should be auto-enabled to add new accounts as they come online. Note that this should be done in every region as a defense in depth measure, with the understanding that the PBMM SCP will prevent service usage in all other regions.

Config

AWS Config provides a detailed view of the resources associated with each account in the AWS Organization, including how they are configured, how they are related to one another, and how the configurations have changed on a recurring basis. Resources can be evaluated on the basis of their compliance with Config Rules - for example, a Config Rule might continually examine EBS volumes and check that they are encrypted.

Config may be enabled at the Organization level - this provides an overall view of the compliance status of all resources across the Organization.

Note: At the time of writing, the Config Multi-Account Multi-Region Data Aggregation sits in the Organization Management (root) account. The AWS Secure Environment Architecture will recommend that this be situated in the security account, once that becomes easily-configurable in Organizations.

Cloudwatch Logs

CloudWatch Logs is AWS' logging aggregator service, used to monitor, store, and access log files from EC2 instances, AWS CloudTrail, Route 53, and other sources. The *AWS Secure Environment Architecture* recommends that log subscriptions are created for all log groups in all workload accounts, and streamed into S3 in the log-archive account (via Kinesis) for analysis and long-term audit purposes.

SecurityHub

The primary dashboard for Operators to assess the security posture of the AWS footprint is the centralized AWS Security Hub service. Security Hub should be configured to aggregate findings from Amazon GuardDuty, AWS Config and IAM Access Analyzers. Events from security integrations are correlated and displayed on the Security Hub dashboard as 'findings' with a severity level (informational, low, medium, high, critical).

The *AWS Secure Environment Architecture* recommends that certain Security Hub frameworks be enabled, specifically:

- AWS Foundational Security Best Practices v1.0.0
- PCI DSS v3.2.1
- CIS AWS Foundations Benchmark v1.2.0

These frameworks will perform checks against the accounts via Config Rules that are evaluated against the AWS Config resources in scope. See the above links for a definition of the associated controls.

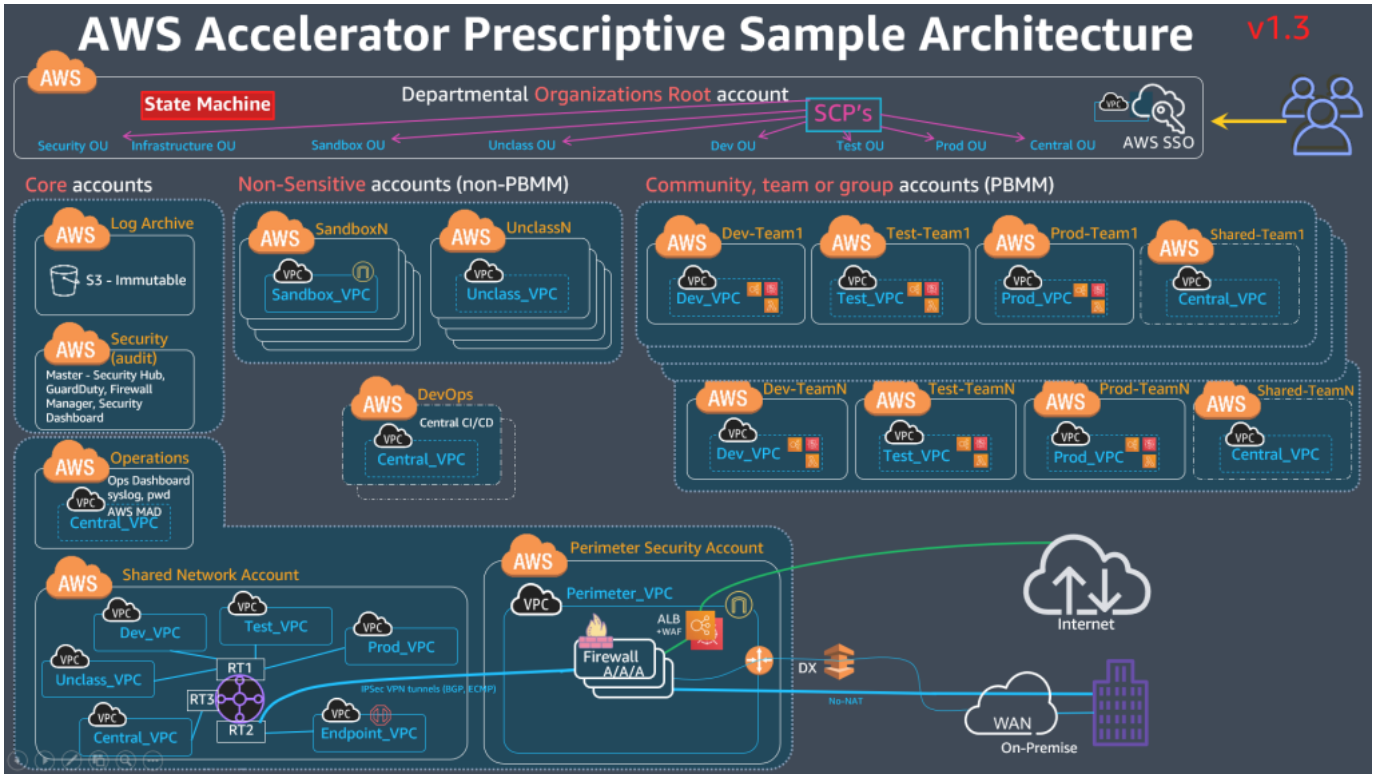
Systems Manager Session Manager

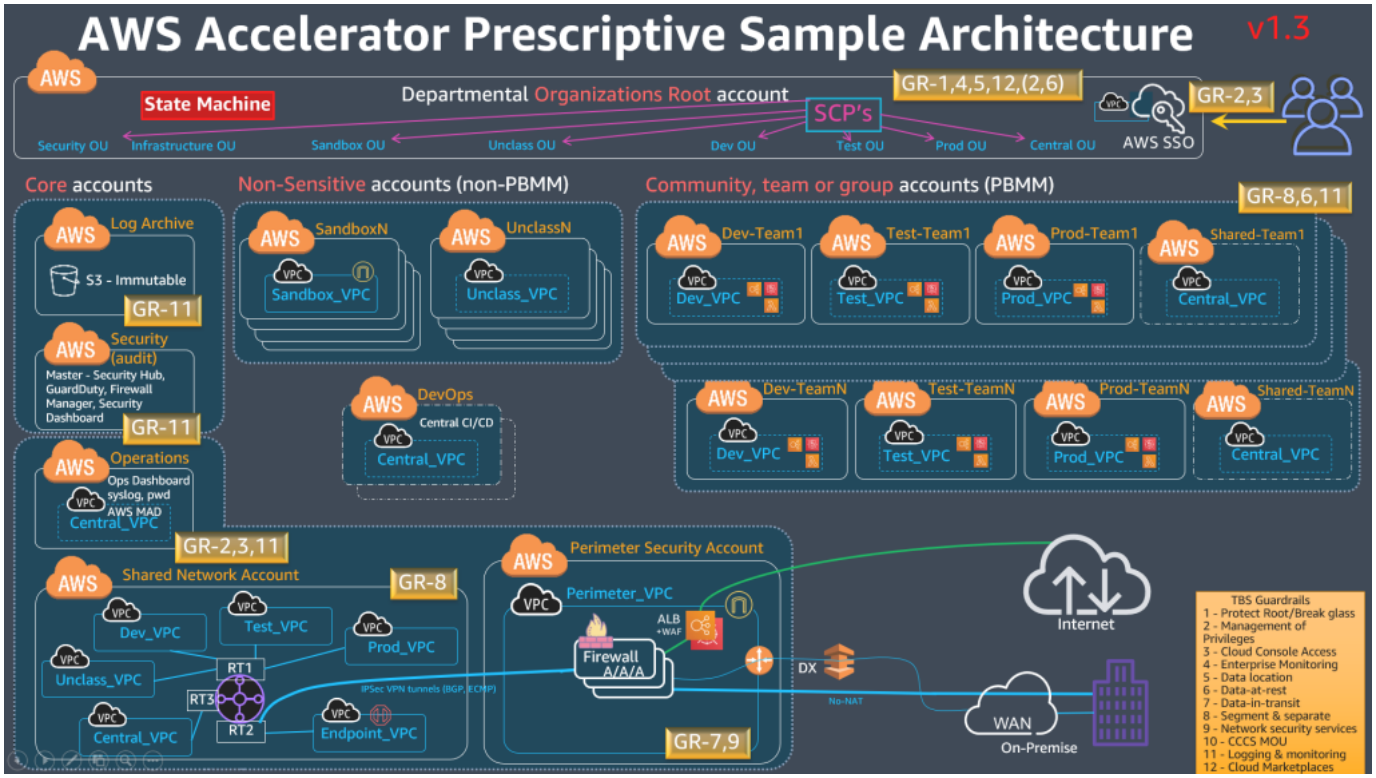
Session Manager is a fully managed AWS Systems Manager capability that lets you manage your Amazon Elastic Compute Cloud (Amazon EC2) instances, on-premises instances, and virtual machines (VMs) through an interactive one-click browser-based shell or through the AWS Command Line Interface (AWS CLI). Session Manager provides secure and auditable instance management without the need to open inbound ports, maintain bastion hosts, or manage SSH keys. Session Manager also makes it easy to comply with corporate policies that require controlled access to instances, strict security practices, and fully auditable logs with instance access details, while still providing end users with simple one-click cross-platform access to your managed instances.[1]

The *AWS Secure Environment Architecture* recommends that you choose to store session log data in a centralized S3 bucket for auditing purposes and encrypt with Key Management Service (KMS). In addition, session log data should also be configured to be sent to Amazon CloudWatch Logs with KMS encryption using your AWS KMS key.

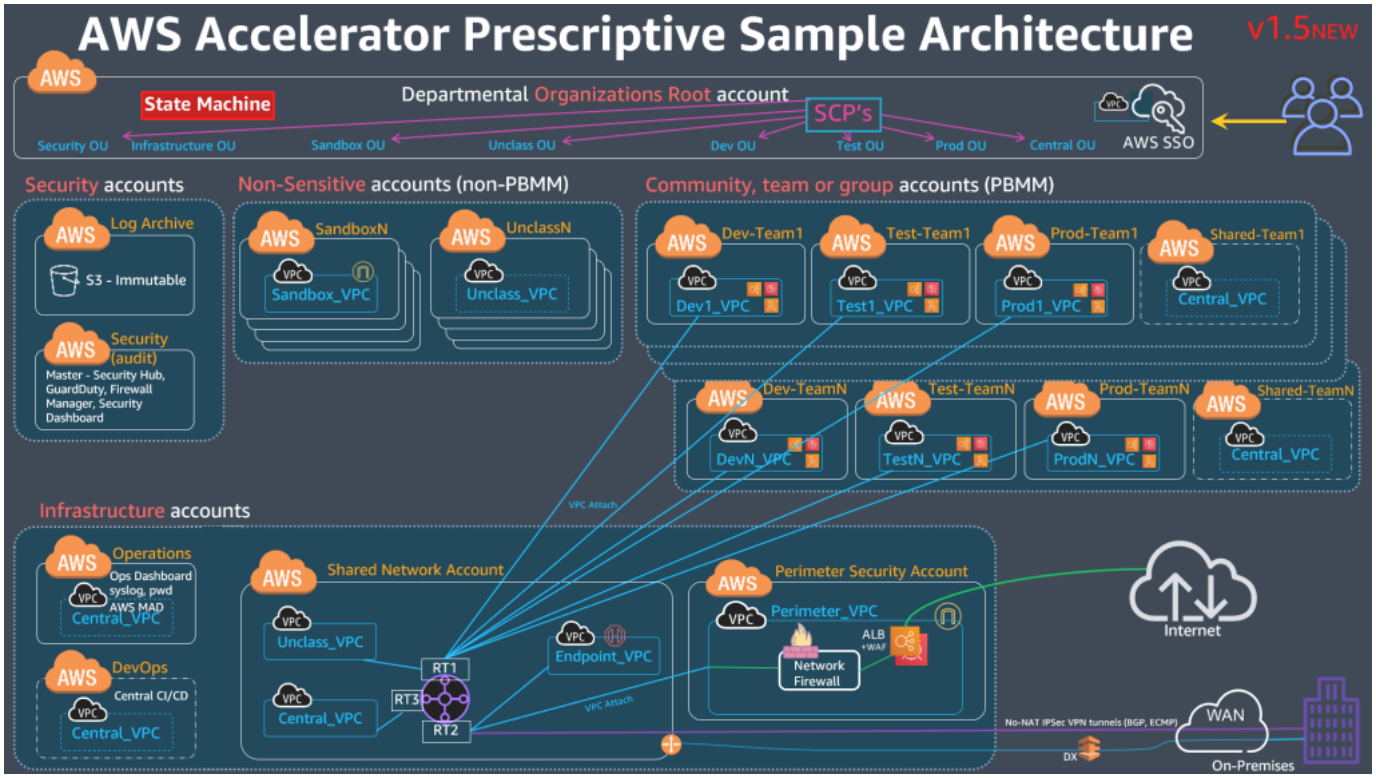
7.1.6 Prescriptive Sample Architecture Diagrams

Shared VPC Architecture

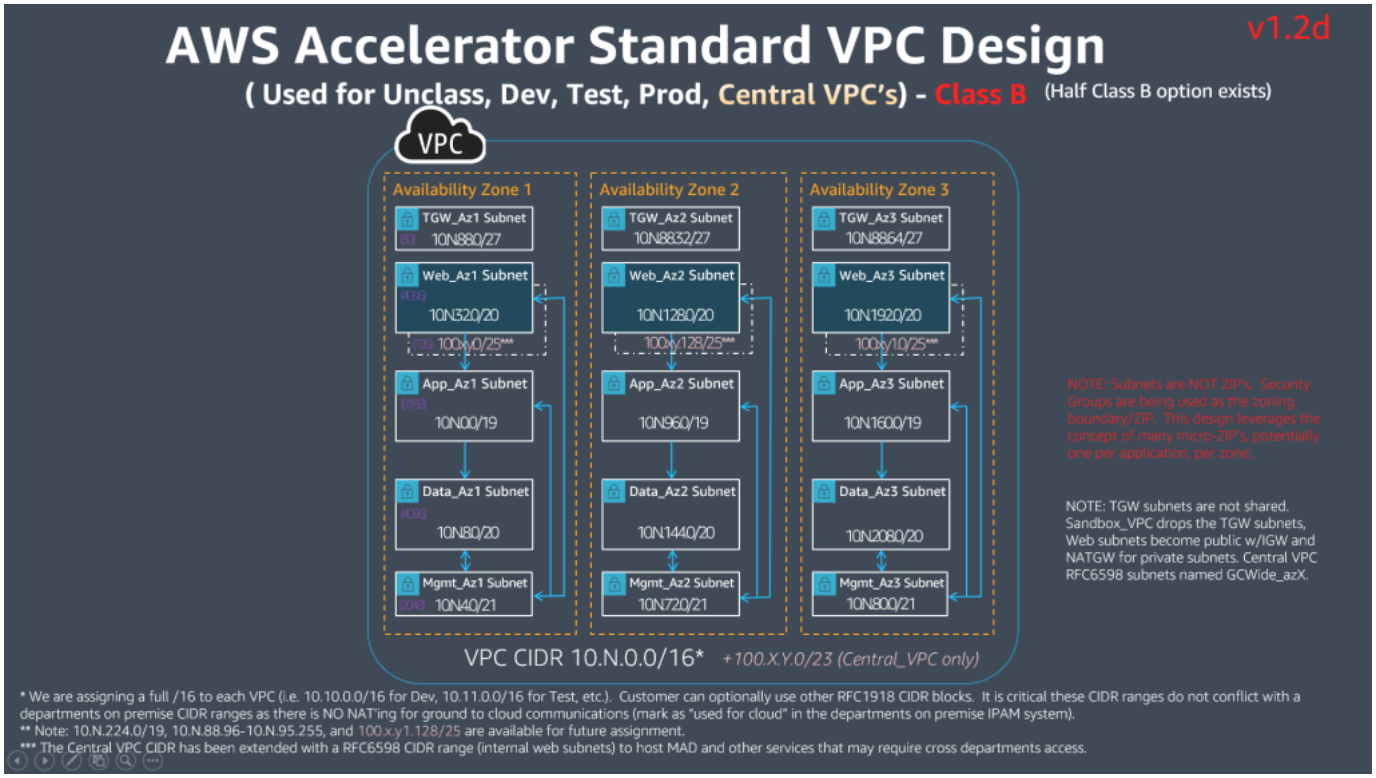


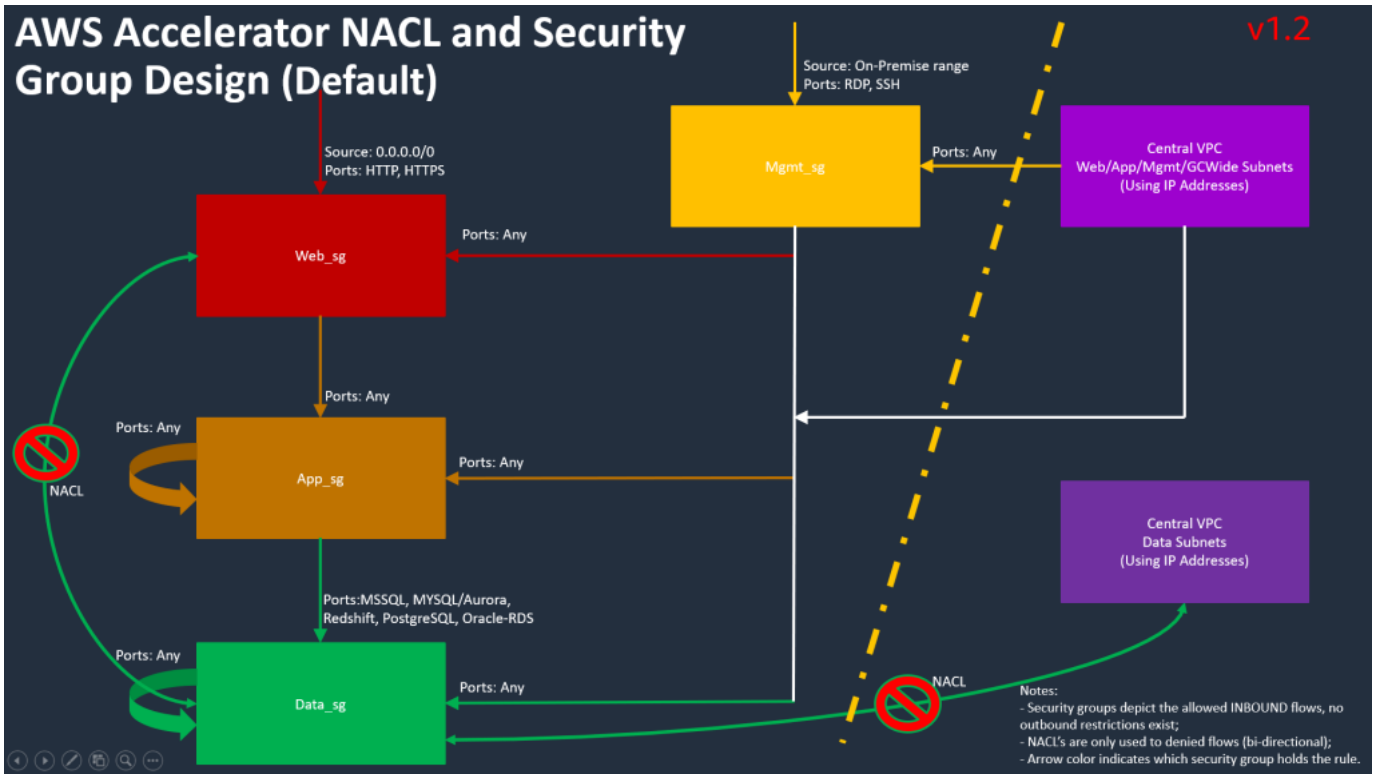


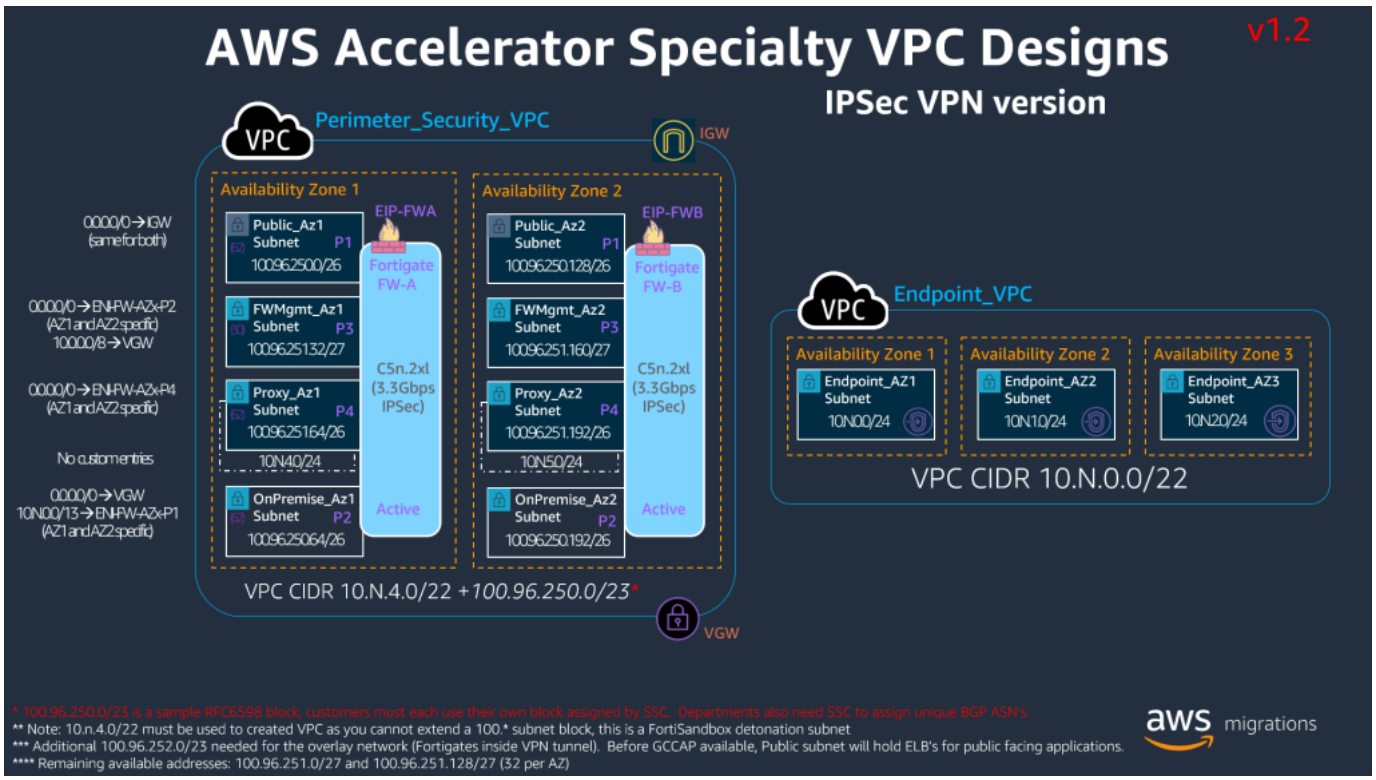
Spoke VPC Architecture



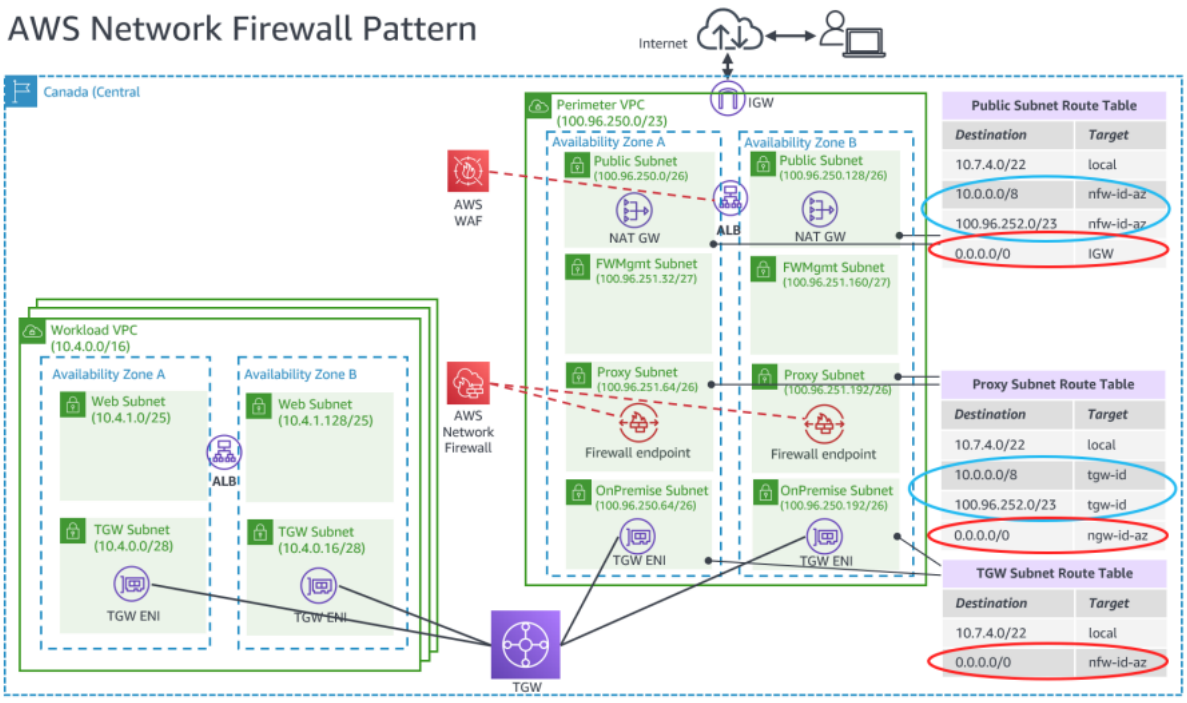
VPC and Security Group Patterns



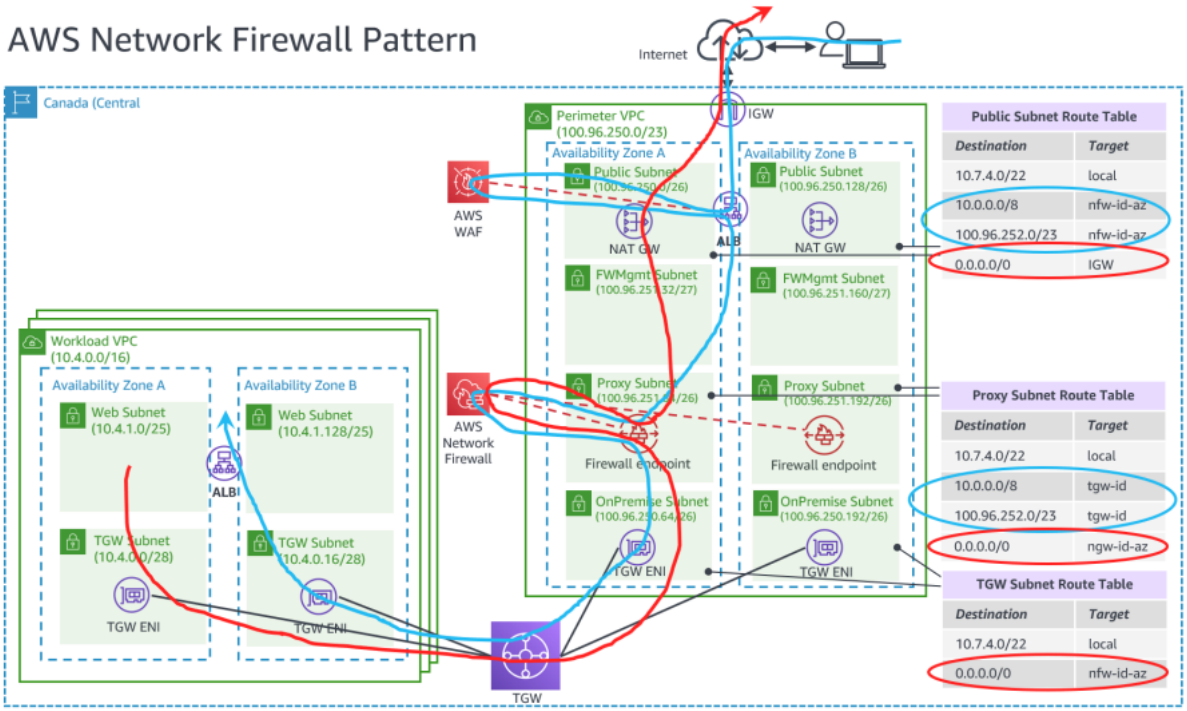

AWS Accelerator NACL and Security Group Design (Default)
v1.2
Source: On-Premise range
Ports: RDP, SSH
Source: 0.0.0.0/0
Ports: HTTP, HTTPS
Mgmt_sg
Ports: Any
Central VPC
Web/App/Mgmt/GCWide Subnets
(Using IP Addresses)
Web_sg
Ports: Any
Ports: Any
Ports: Any
NACL
App_sg
Ports: Any
Central VPC
Data Subnets
(Using IP Addresses)
Ports:MSSQL, MYSQL/Aurora,
Redshift, PostgreSQL, Oracle-RDS
Ports: Any
Data_sg
Ports: Any
NACL
Notes:
- Security groups depict the allowed INBOUND flows, no outbound restrictions exist;
- NACL's are only used to denied flows (bi-directional);
- Arrow color indicates which security group holds the rule.


AWS Accelerator Specialty VPC Designs
v1.2
IPSec VPN version
VPC
Perimeter_Security_VPC
IGW
Availability Zone 1
Availability Zone 2
EIP-FWA
EIP-FWB
Public_Az1 Subnet P1 100.96.250.0/26
Public_Az2 Subnet P1 100.96.250.128/26
Fortigate FW-A
Fortigate FW-B
FWMgmt_Az1 Subnet P3 100.96.251.32/27
FWMgmt_Az2 Subnet P3 100.96.251.160/27
C5n.2xl (3.3Gbps IPSec)
Proxy_Az1 Subnet P4 100.96.251.64/26
Proxy_Az2 Subnet P4 100.96.251.192/26
10.N.4.0/24
10.N.5.0/24
OnPremise_Az1 Subnet P2 100.96.250.64/26
OnPremise_Az2 Subnet P2 100.96.250.192/26
Active
VPC CIDR 10.N.4.0/22 +100.96.250.0/23*
VGW
0.0.0.0/0 → IGW (same for both)
0.0.0.0/0 → ENI-FW-AZx-P2 (AZ1 and AZ2 specific) 10.0.0.0/8 → VGW
0.0.0.0/0 → ENI-FW-AZx-P4 (AZ1 and AZ2 specific)
No custom entries
0.0.0.0/0 → VGW 10.N.0.0/13 → ENI-FW-AZx-P1 (AZ1 and AZ2 specific)
Endpoint_VPC
Availability Zone 1
Availability Zone 2
Availability Zone 3
Endpoint_AZ1 Subnet 10.N.0.0/24
Endpoint_AZ2 Subnet 10.N.1.0/24
Endpoint_AZ3 Subnet 10.N.2.0/24
VPC CIDR 10.N.0.0/22
* 100.96.250.0/23 is a sample RFC6598 block, customers must each use their own block assigned by SSC. Departments also need SSC to assign unique BGP ASN's.
** Note: 10.n.4.0/22 must be used to created VPC as you cannot extend a 100.* subnet block, this is a FortiSandbox detonation subnet
*** Additional 100.96.252.0/23 needed for the overlay network (Fortigates inside VPN tunnel). Before GCCAP available, Public subnet will hold ELB's for public facing applications.
**** Remaining available addresses: 100.96.251.0/27 and 100.96.251.128/27 (32 per AZ)
aws migrations

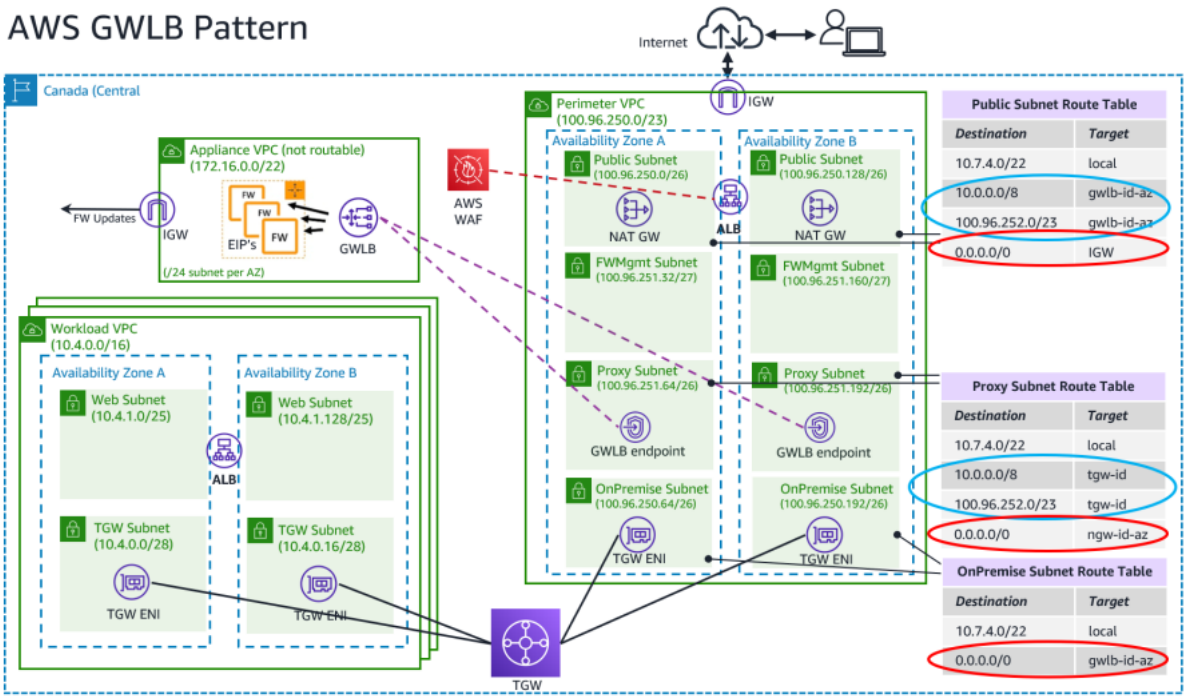
Additional Perimeter Patterns



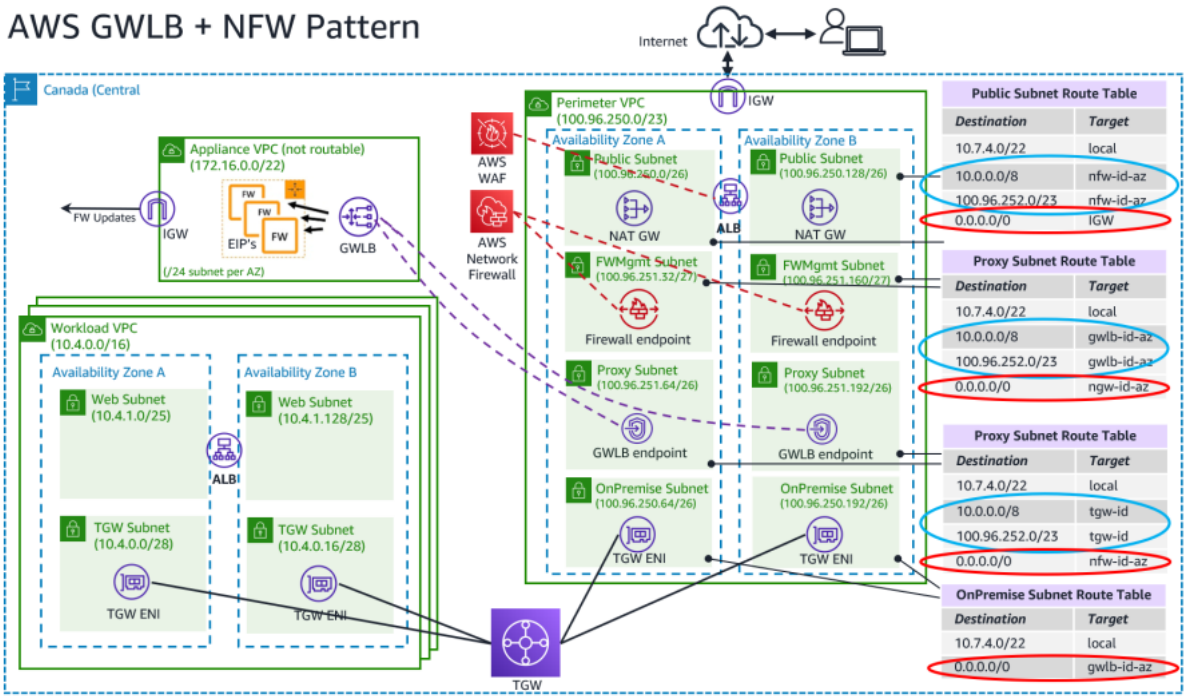
NOTE1: Distinct route tables required per AZ, which targets the local AZ's nfw, gwlb or ngw endpoint



NOTE1: Distinct route tables required per AZ, which targets the local AZ's nfw, gwlb or ngw endpoint



NOTE1: Distinct route tables required per AZ, which targets the local AZ's nfw, gwlb or ngw endpoint



NOTE1: Distinct route tables required per AZ, which targets the local AZ's nfw, gwlb or ngw endpoint

8. ASEA Workshops

8.1 Accelerator Administrator Immersion Day

The Accelerator Administrator Immersion Day workshop is focused on helping administrators who will be administering the landing zone understand how they can design, build and operate the components in ASEA. Click here for an overview of the topics covered.

8.2 Accelerator Workload/Application Team Immersion Day

The Accelerator Workload/Application Team Immersion Day workshop is focused on helping project teams understand what it means to operate within an ASEA managed environment. Click here for an overview of the topics covered.

9. Configuration File Schema Documentation

English Documentation: https://awssea.github.io/schema/en/index.html

Documentation Française: https://awssea.github.io/schema/fr/index.html