VFX Burst Rendering on AWS using ISE Content Security recommendations and best practices

Security Control Mapping

April 2019

AWS UK

John Bennett and Paco Hope, Jonathan Jenkyn, Esteban Hernandez, Mike Owen

Table of Contents

Summary

This document accompanies the AWS Quick Start that uses an AWS CloudFormation template to deploy a framework architecture for Visual Effects (VFX) Burst Rendering on the AWS Cloud.

The document acts a recommendation checklist to assist the VFX studio to prepare for an Independent Security Evaluators (ISE) content security audit of their production Burst Rendering environment. It describes how each ISE best practice and recommendation in the *'Studio Security Controls for VFX/Rendering'* AWS Artifact can be implemented in the AWS Cloud deployed burst rendering environment. These recommendations will assist the VFX studio build up to their content security compliance goals by highlighting the areas that the VFX studio needs to describe and document to be ready for an ISE audit.

The CloudFormation deployed framework architecture does not deliver compliance with each ISE control recommendation 'as-is'. The VFX studio is required to perform further customisation of their deployed burst rendering architecture to build up to their content security compliance goals. This document provide best practice and recommendation assistance.

Data Classification

The sensitivity of the content owner's data and impact of financial loss, negative brand reputation and penalties if leaked or stolen determines the VFX studio's required level of security to work on the content owner's data. We recommend that the following questions should be asked between the VFX studio and the content owner to determine the type of content and value of content to sort the data in to classification levels and thereby determine the VFX studio's approach to implementing many of the ISE recommendations in this document. The tiering classification is typically defined by the content owner and each owner has a varying way of defining them based upon client and project.

- What is the type of the per-production content to be secured?
- What is the value of the per-production content to be secured?
- Classify the content as Tier 1, Tier 2 or Tier 3 content. Examples are:-
    - Tier 1: High-Security Content - Pre-Release Theatrical, Pre-Release Broadcast, Pre-Release Gaming, High Security Advertising
    - Tier 2: Mid-Security Content - Post-Release Theatrical, Post-Release Broadcast, Post-Release Gaming, Mid-Level Security Advertising
    - Tier 3: General-Security Content - General Advertising, Home Entertainment released titles
- Determine how the tiered content type and content value determines the required per-production segregation granularity of AWS accounts, Amazon Virtual Private Cloud (Amazon VPC's) and AWS Key Management Service (AWS KMS) encryption keys to achieve ISE best practice recommendations. Tier 1 content will require this level of granularity. Tier 2/Tier 3 content may not.

- Determine the administrative/cost overhead of using separate per-production project AWS accounts, Amazon VPC's and AWS KMS encryption keys relative to the per-production project asset value, project duration and number of concurrent projects. Tier 1 content will require this level of granularity. Tier 2/Tier 3 content may not.

Controls

The following controls are implementation best practices and guidelines for each ISE control in the *'Studio Security Controls for VFX/Rendering'* AWS Artifact.

1.1 AWS Management Console - Use a separate account for each production, client and render workflow

The supplied AWS CloudFormation template deploys a standardised, framework architecture for VFX Burst Rendering in a specified AWS account. Although the CloudFormation template can be deployed multiple times in different AWS accounts to create segregated, per-production environments within each account, this is not the best way to achieve that goal. We recommend that VFX studios should customise the CloudFormation template relative to their determined tier classification of their data (see above) and further best practices in this document. The tier classification for a production generally determines when separate AWS accounts are required. Tier 1 high-security content will require a separate AWS account.

Use AWS Organizations

Most VFX studios will use several AWS accounts and we recommend organising these multiple AWS accounts using AWS Organizations[1]. Before deploying the CloudFormation template, an Organization Unit (OU) structure should be created[2] that facilitates the centralised management of AWS accounts. The CloudFormation template can then be deployed within a nominated OU member account.

- If the data is Tier 1, then use AWS Organizations to create a dedicated account for it[3] or repurpose an existing AWS account that is not used for anything else.
- For Tier 2/Tier 3 productions, identify the existing AWS account that will be used for the work.
- Create the Organization Units (OU) to group all accounts together for single unit management of master and member accounts.
- Define the relationship between the OU master account and the member accounts required for production work. Create a suitable OU account structure relative to different per-production projects across the enterprise, business line, or business location.
- Create a new member account within the OU structure for every new per-production project.

[1] https://aws.amazon.com/organizations/

[2] https://docs.aws.amazon.com/organizations/latest/userguide/orgs_tutorials_basic.html

[3] https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_accounts_create.html

- Use OU Service Control Policies[4] (SCPs) to enable entities to only use the services allowed by both the SCP and the AWS Identity and Access Management (IAM) policy for the member account.

Use IAM to control access

Use IAM for the segregation of separate roles and their permissions to allow and deny access to specific AWS resources. This enforces the segregation of duties between the roles with full access to the resources and the roles with limited access to the resources, so that only experienced administration roles can make changes to controlled areas of the deployed environment.

Most AWS customers who operate many accounts create their IAM users in a single AWS account, and then use AssumeRole to allow authorised users to take actions across accounts[5].

1.2 AWS Management Console - Use Tags to logically relate and manage production rendering setups

The CloudFormation template illustratively tags the deployed resource with example tags relative to the user definable per-production project name and category of resource. We recommend customising the CloudFormation template's grouping and tagging schema to meet the requirements for your project and costing categorisation.

- Use AWS Resource Groups[6] and Tagging[7] to categorise the groups of resources used for each per-production project assuming that this hierarchy is a combination of being AWS Organizations OU member account related, per-production project name related, AWS Region/AZ related and AWS resource type related.
- Use tagging to add custom attributes such as project name and cost centre to resources such as instances, Amazon Simple Storage Service (Amazon S3) buckets, etc.
- Create a resource grouping and tagging schema that is used to consolidate resources relative to per-production projects.
    - Define the required purpose of tagging and how resource groups and tags can be used to identify per-production project resource types.
    - Define the resource groups relative to illustrative categories of render instance, application instance, asset storage and logging for each per-production project.
    - Define the different resource types within each resource group category. i.e. to illustrate different instance types for different vCPU/memory render needs.
    - Define the default key/value tags for each resource item within its resource group.
- Customise the CloudFormation templates to use the defined resource grouping and tagging schema for the deployment of future resource.
- Define the IAM policy and roles that can create and modify tags.

[4] https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_about-scps.html

[5] https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial_cross-account-with-roles.html

[6] https://docs.aws.amazon.com/awsconsolehelpdocs/latest/gsg/what-are-resource-groups.html

[7] https://docs.aws.amazon.com/awsconsolehelpdocs/latest/gsg/tagging-resources.html

- Consider using AWS Systems Manager[8] to orchestrate the deployment of infrastructure resource groups and the controlled maintenance tasks of applying patches, updates, and configuration changes across the resource groups.

1.3 AWS Management Console - Avoid concurrent access from multiple locations

The CloudFormation template does not deploy an environment that meets this control since it is often required that a user may have valid reasons for having multiple active sessions open for the same user identity. The spirit of this control is that unauthorised access from unexpected locations should be mitigated. We recommend that additional controls can be implemented to mitigate the risk such as the following:

- Create an IAM role policy that denies AWS API calls except from specifically authorised source IP's (e.g. corporate egress IPs, data centre egress IPs) in the AWS environment[9]. Use this role policy for IAM assumed user and service-principal identities.
- As per #1.4, enforce a user inactivity timeout period appropriate to the data tier classification.
- Enable Amazon GuardDuty[10] to detect unusual activity in the AWS environment. Consider creating alerts for UnauthorizedAccess:IAMUser/ConsoleLoginSuccess.B[11] and/or UnauthorizedAccess:IAMUser/MaliciousIPCaller events that might indicate logins from unusual places.

1.4 AWS Management Console - Enforce custom session Inactivity termination

The CloudFormation template deploys an environment with the default session timeout of 12 hours when signing in to the AWS Console with the IAM user credentials. In order to reduce this to a smaller time limit, it is necessary to assume roles. We recommend:-

- Using the assume-role method via the AWS Command Line Interface (CLI) or API which permits the definition of a configurable session timeout for an IAM role and hence a configurable credential lifetime.
- Using an existing Identity Provider (IDP) system (e.g., Microsoft Active Directory) to federate identities from the IDP to AWS. In that case, the session inactivity timeout can be enforced at the master IDP system. The method of identity federation user sessions (see #7.2) between the IDP system and the IAM role uses the assume-role API.

[8] https://aws.amazon.com/systems-manager/

[9] https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_examples_aws_deny-ip.html

[10] https://aws.amazon.com/guardduty/

[11] https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_unauthorized.html#unauthorized4

1.5 AWS Management Console - Enforce Two-Factor authentication for portal access

The CloudFormation template cannot enable multi-factor authentication (MFA) itself. Each IAM user that wants to use MFA will need to provision that access individually. Alternatively, if identity is federated from an existing IDP system (e.g., Okta or Microsoft Active Directory) then the multi-factor authentication is controlled at that IDP instead of in AWS itself. We recommend:-

- First Factor: Password policy
    - IAM permits establishing a password policy[12] with a wide variety of composition rules, rotation requirements, and restrictions.
- Second Factor:
    - AWS supported MFA mechanism[13] should be enabled when using IAM user access. Use either a Virtual MFA device using a time-based on-time password (TOTP) app on the user's smartphone or tablet, or a Physical MFA device such as Gemalto or YubiKey.
    - Access to AWS root accounts should use an AWS supported MFA mechanism[14] regardless of whether identity federation is implemented.
    - Formalise a handling process for access to each master and member root account.
        - Put the root password and MFA token information into an internal escrow service with tight control around audited access.
        - Ensure that the root account is not used for daily usage. Instead create a highly privileged IAM admin role that replaces the use of the root user credential.
        - Enable a controlled and limited list of users to have the necessary permissions to assume the IAM admin role.
    - Verify that IAM users have MFA enabled. Review the IAM credentials report that lists users in the AWS account and the status of their credentials including passwords, access keys and MFA devices. Remove old IAM credential reports (as CSV files) from the AWS console.
    - Verify that the IAM user activity and the IAM admin role activity are logged as AWS CloudTrail Events and that the necessary escalation alarm notification is in place for access discrepancies relative to a known list of identity federated users able to assume the IAM admin role.

1.6 AWS Management Console - Deploy topologies and workloads consistently

The CloudFormation template deploys each service and application stack of the framework burst rendering environment within the specified AWS account. Further customisation of the CloudFormation template is required for integration with the on-premises render environment and the determined granularity between AWS accounts (see #1.1).

[12] https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_passwords_account-policy.html

[13] https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa.html

[14] https://docs.aws.amazon.com/IAM/latest/UserGuide/id_root-user.html#id_root-user_manage_mfa

We recommend that deployment should use an automated, repeatable method using a validated CloudFormation template stack to reduce the chance of individual configuration errors. The authority for deploying resources (e.g., CloudFormation CreateStack) should be independent of the authority to manage those resources (e.g., Windows logins or Linux logins or AWS Systems Manager Session Manager, etc.)

We recommend customising the CloudFormation templates to provide automated deployments of:

- A single management environment integrated with the on-premises environment.
- Multiple per-production project environments each of which is integrated with the single management environment and each of which is integrated with the on-premises environment.

These CloudFormation templates can then be used as the automation method for consistently deploying subsequent per-production project environments.

1.7 AWS Management Console - Limit use of personal accounts for root account

The CloudFormation template deploys into an already existing AWS account and is agnostic of the Enterprise's method of accessing the account. We recommend:-

- Configure the AWS account contact information with a corporate email distribution list in your domain (e.g. *aws-<org_name>@example.com*) and company phone number rather than an individual user's personal email address or personal phone number. The Enterprise needs to self-enforce that the root account does not use an individual user's personal email address.
- As per #1.5 ensure that the root account is not used for daily usage. Instead create an IAM admin role that can be used for daily admin usage and replaces the use of the root user credential.
- The AWS account identity should be secured by enabling and configuring the *Configure Security Challenge Questions* and should be included in the formalised handling process for access to master and member root accounts.

2.1 AWS Networking - Utilise VPC Security Groups

The CloudFormation template uses AWS security groups to control network connectivity between the deployed resources in a per-production project Amazon Virtual Private Cloud (VPC) and in the management VPC, both within the same AWS account. We recommend further customisation of the provided example security groups to further secure inbound and outbound traffic flow on these resources relative to the specific render applications and interaction with the on-premises render environment. We recommend:-

- Delete the default VPC, subnets, routing tables, etc. Delete or avoid using default security groups and default Network ACLs.
- Define security groups to permit allowed traffic between instances in the per-production project VPC and the management VPC, and the on-premises network. Access is denied by default. Details will depend on the Amazon Elastic Compute Cloud (Amazon EC2) instances and their purposes. See #2.5.
- Create a dedicated IAM role for networking administration with the necessary authorisation to perform maintenance on VPC, security group, Network ACLs, AWS Direct Connect and IPsec VPN resources. Generally speaking, ad hoc changes to the AWS environment should not be made. Instead, changes to the CloudFormation template should be made, and then the changes should be deployed as a change set[15].
- Consider creating AWS Config Rules to identify non-compliant resources (e.g., security groups with liberal access, internet gateways attached to VPCs that should not have internet access, etc). These rules can detect, alert, and even automatically correct non-compliant resources.

2.2 AWS Networking - Isolate virtual appliances in their own subnet

The CloudFormation template deploys resources within multi-tiered subnets of the per-production project VPC and the management VPC, within the same AWS account.

By default Amazon EC2 resources are isolated and have no connectivity to anything. Connectivity from an EC2 resource is only possible if a security group and a subnet route permit both permit it.

Connectivity between EC2 resource in the per-production VPC and the management VPC is routed between VPCs via a VPC peering connection. Connectivity between the instances within the private subnets of the VPCs is enabled using security groups and subnet routes via the VPC peering connection.

Connectivity between EC2 resource within the VPCs and other supported AWS services outside of the VPC (e.g., Amazon S3, AWS KMS, etc.) is routed via VPC endpoints[16]. Connectivity between the instances within the private subnets of the VPCs and the supported AWS services outside of the VPCs is enabled using security groups and subnet routes via the VPC endpoints. Traffic via the VPC endpoints and the supported AWS services outside of the VPC does not leave the Amazon network.

[15] https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-updating-stacks-changesets.html

[16] https://docs.aws.amazon.com/vpc/latest/userguide/vpc-endpoints.html

We recommend further customisation of the CloudFormation templates after the initial deployment to control how instances use routable VPC gateway devices (virtual private gateway and internet gateway devices) and VPC endpoint devices for routing resource from within multi-tiered subnets to other VPC's, within AWS Region resource and to on-premises resource. We recommend:-

- Use security groups and Network ACLs to control how the outbound network traffic on limited instances (i.e. the example license server) in the private subnets of the management VPC can route via the public subnet NAT gateway and internet gateway (IGW) to the public internet. Ensure that content-aware instances in the private subnets do not have internet access and do not have default gateways that route to the internet gateway (IGW).
- Use security groups and Network ACLs to control how network traffic on instances in the private subnets of the VPCs can route via the addition of a virtual private gateway (VPG) for private connectivity to the on-premises environment. Use AWS Transit Gateway[17] to create connectivity between multiple per-production project VPCs and the management subnet. Transit Gateway can also facilitate network access across AWS accounts and across VPCs and subnets when EC2 instances need access to each other, but the instances are not in the same AWS account. Security groups, network ACLs, and routing tables can then be used to regulate the level of access between resources in VPCs. For example, Lambda functions, can run in a VPC and therefore be subjected to the VPC networking restrictions.
- Configure Amazon GuardDuty threat detection service to monitor the workflows and analyse events from CloudTrail, VPC Flow Logs and DNS Logs, and when it detects a potential threat it delivers a detailed Finding alert to the GuardDuty console and to Amazon CloudWatch Events. Nominated Findings can then trigger AWS Lambda for granular control of automated remediation or prevention.

2.3 AWS Networking - Apply a multi-tiered architecture for VPC

The CloudFormation template deploys resources within the multi-tiered subnets of the per-production project VPC and the management VPC, within the same AWS account.

- The per-production project VPC contains private multi-tiered subnets which contain the content aware burst rendering resource. The content aware resource in these private subnets routes via the virtual private gateway (VPG) to the on-premises render environment. The private subnets in the per-production project VPC have no access to an internet gateway (IGW).
- The management VPC contains private multi-tiered subnets which contain content aware resources and content unaware resources. They can communicate only with the on-premises environment and the AWS services for which endpoints are created.
- Note the inter-VPC costs for data transfer. Inter-VPC peering traffic flow should be kept to a minimum by ensuring that applications in the management VPC exchange minimal data with the peered, multiple per-production VPCs.

We recommend further customisation of the CloudFormation templates after the initial deployment to deploy an operational burst rendering environment for full account segregation. Each per-production project requires its own AWS account containing it's per-production project VPC. A

[17] https://aws.amazon.com/transit-gateway/

singular AWS account is required for the singular management VPC, but peered to multiple per-production project accounts. We recommend:-

- Segregate content aware and content unaware resources into different multi-tiered subnets in the per-production project VPC and into different multi-tiered subnets in the management VPC.
- Use security groups and Network ACLs to control how the categorised resource within the multi-tiered subnets can route to their VPC gateway devices and VPC endpoints.
    - Per-production project VPC:
        - Define which resource and applications the burst rendering EC2 instances in the private back end subnet and in the private mid subnet need to access in other subnets (same VPC, peered VPC via VPC endpoints, on-premises subnets via virtual private gateway).
        - Note that there is no public subnet.
    - Management VPC:
        - Define which resource and applications the EC2 instances in the private, back end subnet need to access in other subnets (same VPC, peered VPC via VPC endpoints, on-premises subnets via Virtual Private Gateway).
        - Define which resource and applications in the private subnets may require routed access to the internet via the public front end subnet (via the NAT gateway and Internet Gateway), and why they need it. i.e. third party cloud license server for usage-based licensing. i.e. the management consoles of third party storage clusters placed in a private subnet that need third party vendor access for support of the storage cluster.
- As per #2.2 use AWS Config, CloudWatch Logs and GuardDuty to validate operation and provide escalation notification for deviation from defined conditions.

2.4 AWS Networking - Create a separate VPC for productions

The CloudFormation template deploys resources in the specified AWS account (ideally an OU member account) and uses the specified AWS Region. It creates a per-production project VPC and a management VPC. Both VPCs are peered so they can communicate freely, subject to restrictions imposed by security groups, routing tables, and Network ACLs.

We recommend further customisation of the CloudFormation template to deploy an operational burst rendering environment. Each Tier 1 project requires its own AWS account containing per-production VPCs. A singular AWS account is required for the common management VPC. When multiple VPCs are to be connected, AWS Transit Gateway can be used instead of VPC peering to provide the granular control of VPC to VPC routing. This is not accomplished in the provided CloudFormation template, and must be done separately after the environments are deployed. We recommend:-

- As per #2.2 use GuardDuty to monitor VPCs for unexpected network activity and possible security incidents.

2.5 AWS Networking - Limit default Security Group (SG) VPC communications

The CloudFormation template uses security groups to control inbound and outbound access between the deployed resources within the per-production project VPC multi-tiered subnets and within the management VPC multi-tiered subnets, within the same AWS account. The provided CloudFormation template uses non-default, multi-tiered, private subnets within the deployed VPC's, and uses customised security groups between the example instances in these subnets and other within Region AWS resource and with the on-premises render environment.

We recommend further customisation of the provided template, making adjustments to the security groups as required by the render applications to secure traffic between instances.

- AWS accounts will have default security groups and default VPCs when first created. Do not use the default security group for resources. Define new security groups that only allow the network protocols and ports that are required between each instance and the resources it uses.
- Use the #2.1 IAM role and #2.3 multi-tiered subnet architecture to implement security groups to control inbound and outbound network traffic between instances and resource.
- Only use the minimum required access to security groups relative to the default-deny nature of security groups.
- As per #2.2 use AWS Config, CloudWatch Logs and GuardDuty to validate operation and provide escalation notification for deviation from defined conditions.

2.6 AWS Networking - Tightly configure EC2 instances using Network Access Control Lists

The CloudFormation template uses Network ACL rules to control inbound and outbound traffic between the VPC multi-tiered subnets and the VPC gateway devices and the VPC endpoints. This provides a depth of protection by adding an extra layer of security for the VPC's endpoints over and above that offered just by security groups.

Network ACLs are best used to block communications between network segments. They are not used for individual hosts and they do not map one-to-one with security groups or firewall rules. We recommend further customisation of the template to apply Network ACLs as required. Example Network ACLs that might be useful include:

- Preventing Tier2/Tier3 network segments from reaching Tier1 network segments.
- Preventing traffic outside the Tier1 VPC and the on-premises data centre from reaching the Tier1 VPC

2.7 AWS Networking - Do not use deprecated cryptography when configuring IPsec VPN

The CloudFormation template does not deploy the IPsec VPN technology itself because customer-specific details are required for the configuration of those resources. We recommend further

customisation of the provided template to enable secure connectivity using site-to-site IPsec VPNs[18] between the on-premises environment and the VPCs. The VPN connectivity can be over the Internet or via Amazon Direct Connect (DX). Some customers may have satisfactory cost performance using VPNs over the Internet, but most Tier1 and Tier2 customers are recommended to use VPNs over Direct Connect to achieve their required high bandwidth and low latency. The recommendation is to use both private connectivity (i.e. Direct Connect) and encrypted connectivity (i.e. an IPsec VPN) between the on-premises render environment and the burst rendering environment.

- Define the multiple IPsec VPNs required between the VPC resource and the on-premises render environment resource to ensure encrypted-in-transit communication.
- Configure the IPsec VPNs between the virtual private gateways and the customer gateway to use IKEv2[19].

2.8 AWS Networking - Use IPsec VPN when using Amazon Direct Connect

The CloudFormation template does not deploy an AWS Direct Connect (DX) connection via a virtual private gateway because customer-specific details are required for those resources. Many customers will require a Direct Connect for their low latency, private bandwidth and dedicated connectivity between the on-premises render network and their nearest AWS Region location.

The recommendation is to use the VPN requirements from #2.7 and apply to the private connection over Direct Connect to ensure encryption-in-transit.

3.1 AWS Command-Line - Avoid caching session information

The CloudFormation template does not create users in AWS, so this security requirement is one that is recommended to be implemented by the customer themselves as they operate and maintain their AWS environment.

If federated identities are used, then command lines can be authenticated using credentials[20] from the single-sign-on process for the central IDP system (e.g. Microsoft Active Directory).

If identity federation isn't supported or has not been configured, then extra care should be taken to minimise the risk of exposed AWS credentials. Users will be assigned access key pairs consisting of permanent access keys and secret access keys that they will store in their respective `~/.aws/credentials` files on their local systems (laptops, EC2 instances, etc.). General IAM best practices are documented online[21], and recommendations to manage IAM access key pairs are:-

- Force rotation of IAM access key pairs periodically, such as every 60 days and at the end of the per-production project.

[18] https://docs.aws.amazon.com/vpn/latest/s2svpn/VPC_VPN.html

[19] https://aws.amazon.com/vpn/

[20] https://aws.amazon.com/premiumsupport/knowledge-center/adfs-grant-ad-access-api-cli/

[21] https://docs.aws.amazon.com/general/latest/gr/aws-access-keys-best-practices.html

- Use IAM credential reports[22] to identify users that have not rotated their keys within the required period.

4.1 AWS Compute - Setup virtual machines to have authentication using public keys

The CloudFormation template deploys AWS Systems Manager Session Manager (SSM) as the recommended method of accessing the EC2 instances. It uses SSM in preference to Secure Shell (SSH) for access to EC2 instances, so customers are recommended to continue this practice and use SSM to avoid concerns about password-authenticated SSH. Recommendations are:-

- Use SSM for a browser-based interactive shell (not SSH shell) for a named administrative user managing the Linux EC2 instances. This method doesn't require the need to open inbound SSH ports, manage SSH keys, or use bastion hosts.
    - Create an IAM role and policies to control which SSM users can access each EC2 instance, from which source IP range they access, and the administrative actions that the users can perform on the instances via the SSM.
    - Decide the minimal administrative access needed for each instance application.
    - Verify that SSM commands are logged in CloudWatch Logs and that CloudTrail is able to audit which user accessed an instance.
- An alternative method of administrative SSH access to the Linux EC2 instances is to use a bastion instance for SSH access, but this does require the opening of inbound SSH ports and managing SSH keys via bastion hosts. The CloudFormation template does not deploy a SSH bastion instance method of accessing the EC2 instances, but is described here for legacy purposes for use by VFX studio administrators until they are familiar with the recommended AWS Systems Manager Session Manager.
    - EC2 instances require username/password SSH access disabled and instead must use public key authentication for SSH access. Use SSH with an EC2 key pair for a named administrative user managing the Linux EC2 instances.
    - Launch EC2 instances via an instance profile using an Amazon Machine Image (Amazon AMI) as per #4.2. The AMI should have the public key of the named key pair in the `~/.ssh/authorized_keys` of the accessing named administrative user, and have SSH configured to only accept SSH connections from the bastion instance and only run the minimally required list of administrative commands. Supplement this SSH configuration with an instance security group to only accept SSH connections from the bastion instance and Network ACL from the bastion subnet.
    - Use a bastion instance running SSH agent forwarding (avoiding the need to store public keys on the bastion) and with the bastion instance using a security group configured only to accept SSH connections from limited on-premises render environment IP's.
    - Use the private key of the named key pair in the SSH client of the accessing named administrative user.
    - Implement a procedure to rotate the key pair of the EC2 instances every 60 days. This requires the creation and rotation of a new custom AMI containing the new public key. This requires a high level of administrative overhead.

[22] https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_getting-report.html

- Configure EC2 instance SSH commands from syslogs to be logged in AWS CloudWatch Logs or another off-system log e.g., security information and event management (SIEM) to be able to audit which user accessed an instance and the SSH commands that were run.

4.2 AWS Compute - Utilise hardened OS images for instance instantiation

The CloudFormation template deploys example EC2 instances launched from a base Amazon Linux2 (AL2) AMI. Using an AL2 AMI provides a stable, secure and high-performance base execution environment for applications that enable easy integration with other AWS services such as the AWS CLI, EC2 API and AMI tools, Python Boto3 library, etc.

We recommend further customisation of the EC2 instance AMI's to create customised AMI's that contain the required burst rendering software. These AMIs should firstly be hardened to reduce insecure configurations and resource access, and use encrypted Amazon Elastic Block Store (Amazon EBS) root volumes. Burst rendering EC2 instances created from this hardened AMI will then use encrypted boot and data volumes. Recommendations are:-

- Choose a base Amazon AMI that is compatible with the chosen render application and OS dependency. Ideally use the latest Amazon Linux2 (AL2) AMI as the base image that is already loaded with the necessary Amazon drivers.
- Do not import a VM image from on-premises and then convert it into an AMI. This rarely works well because it will not have the necessary Amazon drivers installed, potentially not be security hardened, and potentially not have the latest cycle of security patches for known compromises.
- Use the CIS Benchmark - Securing Amazon Linux guide to customise the base Amazon AMI and create a custom, hardened and secure, reference burst render image AMI.
- Further customise the hardened AMI to create the reference burst render image AMI containing the additional render applications. Use this AMI for instance launch.
- Create an EC2 instance Launch Template to define:
  - The ID of the reference burst render image AMI as the launched image for each EC2 burst render instance.
  - The IAM role to associate with the launched EC2 burst render instance. Render scheduler applications and render applications running on the instance will then use this instance launch role to access resource.
- Alternatively the render applications can be configured after the instance has launched by using configuration management.
- If EC2 Spot instances are being used, define the process that is triggered if instance interruption notification occurs due to capacity being returned to on-demand instances. Clarify how incomplete render output is captured and subsequently dealt with. See #10.9.
- Consider using Amazon Inspector[23] to report on vulnerable software and to scan hosts to identify vulnerabilities.

[23] https://aws.amazon.com/inspector/

5.1 AWS Storage - Create specific IAM account and access key to access to storage account resources

The CloudFormation template deploys a burst rendering environment that uses several different types of storage, including instance root volumes, asset storage and logging storage. The security controls around them depend on which storage service is used and use least privilege identity policies and resource policies to control encrypted access to each storage service.

IAM policies govern access to the storage resources. Those policies are generally attached to roles, and roles are attached to principals (like users, EC2 instances, or AWS Lambda functions). The resource policies define the actions that can be performed on the storage. The attachment of roles to entities defines who can perform those actions. The combination of the identity policy and the resource policy assigned to the IAM role ensures that an entity only has the minimal privileges to access the resource its requires to be operational.

The CloudFormation template deploys examples of these IAM policies and roles. We recommend that VFX studios should customise the CloudFormation template relative to their required policies and roles determined by the tier classification of their data and further best practices in this document.

Recommendations on some basic principles to follow with respect to securing the storage services:-

IAM Roles

- Do not attach significant high privileges to IAM users directly. Do not attach high privilege roles to groups and assign users to these groups. Instead, have users use AssumeRole[24] to assume a role that has high privileges, perform the tasks requiring privileges, and then drop the privileges. This is so those users will not have these high privileges at all times, and only have them when they need them.
- Long-term access keys that do not expire will exist. As mentioned in #9.5, they should be rotated regularly. Do not attach policies that grant access to storage resources to long-lived IAM users or unprivileged roles.
- Do not use store IAM access keys embedded in software applications to access storage resources. It is common to see 3rd party examples where access keys and secret keys are stored in the code, the software environment, or configuration files, etc. This approach creates significant security risk and shouldn't be done for production work.
- If an application running on an EC2 instance needs access to S3 storage, grant the permissions in an IAM policy, assign the IAM policy to a role, and assign the role to the launch of the EC2 instance. The instance then uses temporary credentials via AWS Security Token Service (AWS STS) that automatically auto-expire and auto-renew within the AWS STS service. No manual key management is needed.
- Create an encryption storage access strategy to define the granularity of storage access relative to different use categories of storage types (root, asset, logging) within each per-production project account or common management account, and the individual encryption keys (see #9.1) required for this access granularity.

[24] https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use.html

Storage Types

- **Instance root volume on encrypted EBS:** EC2 instances require access to EBS encrypted root volumes for their launch AMI. By default the root volume of an EC2 instance will be deleted when the instance is terminated. While it is reasonable to assume that render instances are not persistent throughout the project lifecycle, and are initialised and terminated as necessary, it is recommended to use encrypted EBS root volumes to ensure encrypted boot and data volumes.
- **Asset storage repository in S3**: The burst rendering EC2 instances do not directly access asset data on S3 asset storage repository. This S3 storage is used as an asset storage repository and assets are sync'd with the asset storage cache used directly by the burst rendering EC2 instances.
- **Asset storage directly used by the burst rendering EC2 instances:**
  - Primary, on-premises storage only if the network latency between the render instance and the on-premises storage is low enough for direct NFS mounts to function.
  - Persistent asset storage cache for syncing assets from the S3 asset storage repository or for syncing assets directly from on-premises storage.
  - Amazon native encrypted storage - EFS or Amazon FSx for Lustre[25] backed by its own encrypted S3 bucket.
  - Third party parallel file system that uses encrypted, persistent EBS storage volumes with provisioned IOPS as their clustered storage solution.
  - Third party parallel file systems that uses encrypted, persistent NVMe SSD instance store volumes backed by its own encrypted S3 bucket, to deliver high I/O performance as their clustered storage solution.
  - Additionally attacheded EBS volumes. By default additionally attached EBS volumes of a burst render EC2 instance will NOT be deleted when the instance is terminated. The data persists after the instance termination.
- **Logging storage:** CloudTrail logs are encrypted during transfer and at rest (using server-side encryption), and stored in S3 buckets.

Storage Access

- Define the render application's requirements for accessing its asset storage and which type of asset storage it uses.
- Define the on-premises method of copying asset data to the required asset storage types for the render application to then use. Define how the pre-fetch of asset data from the on-premises storage to the asset storage types is handled.
- Create a least privilege resource-based policy for each storage type (instance root volume, asset storage, logging storage). As an example for S3, create an S3 bucket resource policy that defines who can access the S3 bucket and what actions can be performed.
- Assign the resource-based policy and the identity-based policy to the IAM role used to access each storage type. Ensure that the IAM role also has the necessary resource-based policy to access the account-based KMS as per #9.1.
- Clarify which applications can assume which IAM roles using storage resource-based and identity-based policies.

[25] https://aws.amazon.com/fsx/lustre/

5.2 AWS Storage - Periodically update access keys

As per #5.1, we recommend that applications running on EC2 instances use IAM roles with least privilege resource-based policy access and identity-based policy access to the storage types instead of IAM user keys. The application running on the EC2 instance assuming the IAM role then uses temporary credentials via AWS Security Token Service (AWS STS). These credentials automatically auto-expire and auto-renew within the AWS STS service, so no manual user key rotation is needed.

5.3 AWS Storage - Monitor Storage Activity

When sensitive assets are stored in S3, we recommend turning on S3 API logging[26] in order to capture unauthorised access to data objects in S3. This will add additional logging to an existing CloudTrail. In order for the information to be useful, the CloudTrail must be ingested by some sort of monitoring/logging system such as a security information and event management (SIEM) and the SIEM must be configured to alert/respond on unauthorised data access.

Another valuable monitor for S3 activity is AWS Config Rules, which can be used to monitor the permissions[27] of S3 buckets and their S3 objects. It can either alert or simply revert the permissions if an object is made public. The recommended approach is to write AWS Config Rules that repair the problem, rather than send an email to a person who must then repair the problem manually.

6.1 AWS Batch - Periodically update access keys

The CloudFormation template does not deploy AWS Batch as a burst rendering environment since most VFX studio customers use a dedicated render scheduler (such as Deadline in #10) to distribute render jobs on burst rendering EC2 instances.

If a customer chooses to use AWS Batch for rendering, we recommend using IAM roles[28] that grant the AWS ECS instance privileges (e.g., access to read and write relevant S3 buckets) that the batch workloads will require while running. The credentials granted in this way will be temporary and will not require manual rotation or updating.

6.2 AWS Batch - Sanitise and destroy Compute Environment when no longer needed

The CloudFormation template does not use AWS Batch to deploy a burst rendering environment. If customers do use AWS Batch, the compute resources will be automatically reclaimed when the jobs

[26] https://docs.aws.amazon.com/AmazonS3/latest/dev/cloudtrail-logging.html

[27] https://aws.amazon.com/blogs/security/how-to-use-aws-config-to-monitor-for-and-respond-to-amazon-s3-buckets-allowing-public-access/

[28] https://docs.aws.amazon.com/batch/latest/userguide/instance_IAM_role.html

complete. If the EBS volumes of the EC2 instances are encrypted using EBS volume encryption, then the data remains protected until the disk blocks are reclaimed. We recommend:-

- To facilitate the repeat consistency of a per-production AWS Batch compute environment, the setup and closedown of each AWS Batch compute environment should be automated by using create and close scripts run via the IAM create/destroy role in #6.1
- Close down the dedicated AWS Batch Compute Environment at end of each per-production project.

6.3 AWS Batch - Create storage exclusively for specific batch workflows

The CloudFormation template does not use AWS Batch to deploy a burst rendering environment. If customers do use AWS Batch, the EBS volumes connected to the EC2 instances will be dedicated and useable only by those instances.

6.4 AWS Batch - Use security-validated Docker container for batch workflow

The CloudFormation template does not use AWS Batch to deploy a burst rendering environment. If customers do use AWS Batch we recommend performing the following steps to ensure that the Docker container image is sufficiently hardened prior to launching a Batch job.

- Build a Docker Container Image for the AWS Batch ECS environment that has been 'hardened' using the guidelines in #4.2. Start from the default Amazon ECS-optimised AMI to ensure it can run on the managed ECS compute instances.
- Enable the programmatic IAM role for authorised access to the Docker Container Image.
- Use the Job Definition, Container Properties, 'image' parameter to specify the Docker Container Image.
- Use IAM policies and/or monitoring of CloudTrail logs to prevent AWS Batch Job Submission using unapproved Docker Container Image for the job.

6.5 AWS Batch - Use integrity checks on Docker images for batch workflow

The CloudFormation template does not use AWS Batch to deploy a burst rendering environment. If customers do use AWS Batch, the Amazon Elastic Container Registry (ECR) can be used to store the container images that will be deployed.

The SHA256 hash of every container image is available through the ECR API. In order to ensure that the correct container image is used, we recommend that customers maintain a manifest of hashes of approved container images. By querying the metadata about a container image through the ECR, one can determine if the current container image in the registry is the same as the container image that has been approved. We recommend that customers build their own logic as part of deploying container applications that verifies that the hash matches an expected/approved value, and abort the job if a the container image is not an approved image.

6.6 AWS Batch - Hold workflow if Batch applications fail

The CloudFormation template does not use AWS Batch to deploy a burst rendering environment. If customers do use AWS Batch, we recommend monitoring the AWS Batch job state changes asynchronously using AWS Lambda as a CloudWatch Events target. We recommend that customers write their own logic by scheduling Lambda functions to run periodically (every few minutes) using CloudWatch Events:-

- Call AWS Batch DescribeJobs[29] for AWS Batch Job states[30] of RUNNING, SUCCEEDED and FAILED.
- If jobs are in the FAILED state, use the DescribeJobs API operation to gather more information about the job.
- Analyse the retry number for the AWS_BATCH_JOB_ATTEMPT environment variable.
- If the job has failed and already been retried once, terminate the job and raise an alert as an Amazon Simple Notification Service (SNS) message to prompt a manual investigation.

6.7 AWS Batch - Log Batch events for monitoring and diagnostics

The CloudFormation template does not use AWS Batch to deploy a burst rendering environment, but it does deploy standard CloudTrail and AWS Config in the AWS accounts. If a customer uses AWS Batch to deploy their rendering environment, the CloudTrail and AWS Config logs will be available to assist in monitoring and logging events in the environment.

We recommend validating that the monitoring and logging strategy for AWS Batch provides the granularity of logging, monitoring and escalation required for the particular project being rendered. Key operational metrics (compute capacity, running, pending, completed jobs, etc) for AWS Batch jobs can be viewed in the AWS Console or queried through the AWS Batch API.

- Logs for AWS Batch jobs are written to CloudWatch Logs. Setup CloudWatch Events[31] that create alarm events for escalation.
- Define the CloudTrail Batch activity that should create alarm events for escalation.
- Define who is notified of AWS Batch escalation events.
- Use the methods described in #8.1 to monitor, log and escalate AWS Batch activity.

[29] https://docs.aws.amazon.com/batch/latest/APIReference/API_DescribeJobs.html

[30] https://docs.aws.amazon.com/batch/latest/userguide/job_states.html

[31] https://docs.aws.amazon.com/batch/latest/userguide/batch_cwet.html

7.1 AWS Role-Based Access Control - Assign custom roles to manage access to automated users

The CloudFormation template does not deploy render scheduler applications or automation workflow tools, but does provide illustrative example instances on which these applications can be run.

While VFX studios have multiple parallel workflows involving a myriad of artists and resources, the artists and collaborators rarely need direct access to the infrastructure. Their access is largely indirectly controlled within the rendering software and scheduling applications.

We recommend that artists and collaborators should have no direct access to AWS (in the form of user name, password, or access keys) and should only have access mediated by the rending scheduler and other studio-specific applications. If it is necessary to grant access to a third party to launch or manage infrastructure in an AWS environment (e.g., to start or stop jobs, to upload or download content, etc.) we recommend the following guidelines. Each situation will be different, though, and the actual implementation should be customised with respect to the needs of the project.

- Create custom 'service principal' IAM roles that will be used for specific activities.
- Only when absolutely required, due to software limitations, create IAM users that represent services.
- Create IAM policies that grant the fewest privileges to the required resources needed by each burst rendering environment application or automation toolset.
- Create privilege policies for access to specific resources or groups of resources.
- Attach policies to the IAM roles where possible, and to the IAM users where necessary.
- Use IAM Access Advisor[32] to scope down permissions for 'service principal' IAM users to better adhere to the principle of least privilege permissions to perform a specific task. Use the service last accessed data to identify unnecessary role permissions to AWS services and remove permissions for unused services.

7.2 AWS Role-Based Access Control - Extend on-premises identity management to manage access cloud services

The CloudFormation template does not use a corporate Identity Provider (IDP) system to deploy federated authentication to IAM users because customer-specific details are required for the configuration.

Customers who want to enable user identity federation between their existing IDP system (i.e. Microsoft Active Directory) and IAM users will need to follow integration guidance[33]. We recommend that users should be provisioned via the application software (e.g., rendering scheduling

[32] https://aws.amazon.com/blogs/security/automate-analyzing-permissions-using-iam-access-advisor/

[33] https://aws.amazon.com/blogs/security/how-to-set-up-sso-to-the-aws-management-console-for-multiple-accounts-by-using-ad-fs-and-saml-2-0/

software) assuming IAM roles rather than directly at the AWS infrastructure level. As few AWS identities and roles should be created as possible.

- The existing IDP system will integrate with IAM via Security Assertion Markup Language (SAML). This enables users already authenticated within the enterprise to have defined identities in the infrastructure without re-authentication. As per #1.5, root access to master and member accounts requires additionally using an AWS supported MFA mechanism.
- Use the existing IDP system for users who need access to create and manage cloud infrastructure.
- Map existing groups and/or roles in the existing IDP system to roles and privileges in AWS.
- Define which IAM roles can be assumed by which identity federated IAM users.
- Periodically audit the privileges associated with roles in AWS that correspond to roles in the existing corporate directory. Restrict the level of privilege granted to AWS roles to the minimum necessary to perform job duties.

8.1 AWS Monitoring - Monitor and Log Events, Log IP traffic and API Calls

The CloudFormation deploys CloudWatch Logs and CloudTrail to monitor the burst rendering environment. Recommendations for the monitoring and logging of resources:-

- Amazon CloudWatch:
    - Use CloudWatch to collect monitoring and operational data (metrics, events, logs) from the utilised AWS resources and services within the per-production project account and the management account. Monitor trends and perform historical analysis to fine tune resource utilisation. Create dashboards of critical resource metrics and messages.
    - Use CloudWatch Logs to collect and store logs from the utilised AWS resources and services within the AWS account.
    - Configure the delivery of CloudWatch Logs into an encrypted S3 bucket.
    - Use CloudWatch to correlate metrics and logs to diagnose issues and their root cause.
    - Define the CloudWatch metric threshold value changes that should create alarm events for escalation.
    - Use CloudWatch Events to define rules to indicate AWS resources and services events of interest and automated actions to take when a rule matches an event. i.e. a notification escalation action via an Amazon Simple Notification Service (SNS) topic or trigger an automated remediation action via AWS Lambda logic.
    - Use CloudWatch Alarms to set a threshold on a metric and trigger a notification escalation alarm or trigger an automated remediation action.
- CloudTrail:
    - Use CloudTrail to capture AWS resource and service activity events within the AWS account across the used regions.
    - Use CloudTrail to log data events (for the resource data plane operations) and to log management events (for the resource control plan operations) on the utilised resource and services.
    - Define the CloudTrail activity that should create alarm events for escalation.

- Use an AWS Lambda invoked by the logging S3 bucket to process CloudTrail logs and trigger a notification escalation alarm or trigger an automated remediation action.
- GuardDuty:
    - Configure GuardDuty threat detection service to monitor the AWS accounts for malicious or unauthorised behaviour and signs of risk to detect anomalies in accounts and workflows. GuardDuty analyses events from CloudTrail, VPC Flow Logs and DNS logs and when it detects a potential threat it generates CloudWatch Events, which can trigger remediations or be viewed in the GuardDuty web console.
- AWS Config:
    - Use AWS Config to monitor and record resource configurations and automate the evaluation and remediation of deviations from the expected security baseline.
    - Customise pre-built rules to evaluate resource configurations and configuration changes.
        - Correlate configuration changes via CloudTrail Events to obtain details of the event that invoked the change.
        - Correlate configuration changes via AWS Systems Manager for EC2 instance changes and infrastructure changes.
    - Define configuration changes that trigger functions that notify staff to respond or ideally fix issues.

A recommended practice is to establish a separate AWS account for centralising the logging and monitoring of multiple AWS accounts into a single, centralised logging and monitoring account. The CloudFormation template does not deploy that way for deployment simplicity. Further customisation of the CloudFormation templates is required after the initial deployment to achieve this recommendation:-

- See the Center for Internet Security (CIS) AWS Foundations Benchmark Quick Start CIS benchmark on AWS for how to deploy a centralised logging architecture.
- Integrate CloudWatch Logs and CloudTrail logs with a third-party security information and event management (SIEM) software product. Enable the SIEM product to access the account-based encrypted S3 buckets for CloudWatch Logs and CloudTrail Logs.

9.1 AWS Key Management Service (KMS) - Use separate Customer Master Key (CMK) for each production

The CloudFormation template uses the AWS Key Management Service (AWS KMS) to create and control customer master key (CMK) encryption keys used for encrypting data at rest within the specified AWS account. The CloudFormation template creates three separate CMK's managed in KMS using AWS managed keys for simplicity:

- CMK for encrypted EBS root volumes for EC2 instances.
- CMK for encrypted production content assets on EBS volumes, on EFS storage, on S3 storage.
- CMK for encrypted CloudTrail logs on S3.

We recommend the use of CMK’s managed in AWS KMS as customer-managed keys instead of as the AWS managed keys provided in the template. Using customer-managed keys provides more

flexibility to create, disable, and define access controls since the default keys created by AWS managed keys cannot be managed with specific policies.

The three different CMKs used for the different categories of storage (operating system, content assets, infrastructure logging) enable segregation between encrypted resource across AWS accounts. It also provides access controls and additional logging accountability for access to sensitive resources. We recommend:-

- Determine the Data Tiering Classification and whether the new production requires a separate set of CMKs per account to ensure segregated resource encryption for each per-production project account.
  - Tier 1 high-security content will require a separate set of CMKs for each per-production project member account. A separate CMK for encrypted production content assets on EBS, EFS and S3 storage is recommended for each Tier 1 production.
  - Tier 2/Tier 3 content may share CMKs as determined by the confidentiality and separation requirements for the individual products.
- Use the root volume CMK for encryption of the EBS root volumes of the instances. Create a base AMI[34] for each instance type required (compute, rendering, etc.) and encrypt the AMI's root volume using the appropriate CMK. The instances launched from that AMI will have their root volumes encrypted.
- Use the content asset CMK for encryption of content assets on EBS volumes of the file servers (e.g., third party parallel file systems). Specify the content asset CMK during the creation[35] of the volume. This can be done interactively in the web console, from the AWS command line, or in a scripting format like CloudFormation.
- Use the content asset CMK for encrypting content in EFS storage. Specify the CMK by its ARN when the EFS filesystem is created[36].
- Use the content asset CMK for encrypting assets stored in S3. Apply a bucket policy as described in the documentation[37] that requires content to be encrypted with the content asset CMK.
- Use the logging CMK for encryption of logs in S3 logging buckets. The bucket policy specifying the logging CMK must be applied to the logging bucket. CloudTrail must be configured to encrypt logs[38] as they are delivered.

9.2 AWS Key Management Service (KMS) - Use KMS permissions to manage access

Managing access via AWS KMS involves two considerations; which entities can manage the permissions on keys themselves, and which entities can use the keys to gain access to the data. In order to use AWS KMS permissions to manage access, the customer must manage both of these permissions.

[34] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIEncryption.html

[35] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-creating-volume.html

[36] https://docs.aws.amazon.com/efs/latest/ug/encryption.html

[37] https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingKMSEncryption.html

[38] https://docs.aws.amazon.com/awscloudtrail/latest/userguide/encrypting-cloudtrail-log-files-with-aws-kms.html

The CloudFormation template deploys an IAM role with an IAM policy that grants permissions to use AWS KMS. AWS KMS also has a resource-based policy to specify which roles have access to AWS KMS and what actions they can perform. The combination of the identity-based policy and the resource-based policy assigned to the IAM role should be used to ensure that the IAM role only has the minimal privileges to access AWS KMS it requires to be operational.

We recommend the further customisation of the IAM role(s) identity-based policy permissions and the AWS KMS resource-based policy permissions to control how each customer master key (CMK) can be used for the different use categories (root volume, content asset, logging) of encryption.

- Use the data encryption strategy to define the granularity of storage encryption relative to different use categories of storage (root volume, content asset, logging) within an account and the different CMK's required for this granularity.
    - Define the IAM role and resource permissions needed for administrative management of AWS KMS and creation and control of the per-production workflow CMK's.
    - Define the IAM roles and resource permissions needed for accessing the AWS KMS managed per-production project CMK's to be able to generate encrypt/decrypt data keys to access encrypted data on the different use categories of storage (root volume, content asset, logging) within an account.
- Use AWS KMS Grants to provide temporary permissions or more granular permission management for delegating the use of CMKs to other roles.
    - Use Grants for tight control of 'service principal' programmatic access.
    - Access to encrypted content assets on storage in accounts outside the burst rendering per-production project account (i.e. in the content owner's own AWS accounts) can be provided by IAM Cross Account Access. The content owner creates an IAM role in their AWS account with access to their content asset CMK with permissions only to decrypt the assets. The content owner would then use IAM Cross Account Access to grant access to the per-production project account to use the content owner's IAM role to access assets on storage in the content owners AWS account.
- Use CloudTrail to audit AWS KMS key usage by reviewing the log files that contain a history of AWS API calls and related events for the account.

9.3 AWS Key Management Service (KMS) - Segregate Data and Key/Secret Owners

The CloudFormation template uses an IAM role with an identity-based policy to define role permissions to use AWS KMS. AWS KMS also has a resource-based policy to specify which roles have access to AWS KMS and what actions they can perform.

We recommend further customisation of the IAM role(s) to control which roles can manage or use each customer master key (CMK) for the different use categories (root volume, content asset, logging) of encryption.

- Using the IAM role and resource permissions in #9.2 implement a separation of duties approach between IAM roles with AWS KMS permissions for the creation and management of AWS KMS resource, and IAM roles only with AWS KMS permissions for the use of AWS KMS CMKs. Example roles are:-

    - An Admin IAM role with permissions to AWS KMS for the creation and management of AWS KMS resources and CMKs.
    - A Data Operator IAM role with permissions to use AWS KMS resources with privileges to use the content asset CMK used on the per-production storage volumes.
    - A Logging IAM role with permissions to AWS KMS to use the logging CMK to access the encrypted S3 buckets used for CloudWatch and CloudTrail logs.
    - Programmatic access (with use of AWS KMS Grants granularity) will be required by the render application on the EC2 instances. The EC2 render instances need to use an IAM launch role with permissions to use AWS KMS resources with privileges to use the asset CMK used on their attached per-production storage volumes.
    - Programmatic access (with use of AWS KMS Grants granularity) will also be required by third party parallel file system accessing encrypted data on its own EBS storage volumes and accessing encrypted data on alternate storage that it then caches.
- At the end of per-production project unnecessary users should be un-authorised to assume the IAM roles for access to the per-production project AWS KMS CMK's.
- Use CloudTrail to audit AWS KMS key usage by reviewing the log files that contain a history of AWS API calls and related events for the account.

9.4 AWS Key Management Service (KMS) - Audit all Key Management Activity

The key activity that takes place in AWS KMS (e.g., decryption, encryption, permissions changes, etc.) is logged in CloudTrail and can be audited. The information includes the request that was made to AWS KMS, the IP address from which the request was made, who made the request, when it was made, and additional details. For auditing purposes we recommend:-

- Create a CloudTrail Trail to enable continuous delivery of AWS KMS CloudTrail Events to an encrypted S3 bucket using the logging customer master key (CMK).
    - Note that this logging CMK is not the same as the content asset CMK. Hence logging reviewer roles and escalation software applications will only need the necessary IAM role permissions for just this KMS logging CMK.
- Define the CloudTrail AWS KMS event activity that should trigger escalation alarms.
- Monitor and track AWS KMS CloudTrail Events, flagging AWS KMS CloudTrail management events for critical per-production AWS KMS asset keys.

9.5 AWS Key Management Service (KMS) - Periodically Rotate Keys

The CloudFormation template uses AWS KMS with the automatic rotation of customer master keys (CMKs) as a managed service every 365 days to maintain the seamless encryption and decryption of data using the original CMKs and the rotated CMKs.

The encryption in AWS is "envelope encryption"[39] which means that a unique data key is created for each data element that is encrypted. The customer master keys encrypt the data keys, and the data keys encrypt the data.

Manually rotating data keys is not required or recommended since each key is unique. It is arguable whether there is security value in rotating CMKs. If regular CMK rotation is required, then AWS KMS can perform this rotation automatically[40]. It will also maintain prior versions of the key material, so that data encrypted using older versions of the key can still be decrypted.

[39] https://docs.aws.amazon.com/kms/latest/developerguide/concepts.html#enveloping

[40] https://docs.aws.amazon.com/kms/latest/developerguide/rotate-keys.html

10.1 AWS Deadline - Setup Dedicated AWS Account for Deadline

The render scheduler is a licensed and high cost application normally used in a rendering environment where the same, single render scheduler dispatches render jobs across multiple concurrent per-production projects. It is the functionality of the render scheduler itself that defines the ability of a single render scheduler to segregate production assets between concurrent, segregated per-production project environments. Many render schedulers do not yet have the application functionality to distribute renders concurrently at the granularity of a segregated resource in different per-production project AWS accounts and Amazon Virtual Private Cloud (Amazon VPCs).
This CloudFormation template deploys an example, generic render scheduler instance in the management Amazon VPC, and example burst render Amazon EC2 instances in the per-production Amazon VPC, both within the same AWS account. This illustrates how a single render scheduler can fit into a segregated per-production project environment, assuming that the render scheduler functionality permits this level of segregation. The CloudFormation template does not deploy a render scheduler application (i.e. Deadline) on the render scheduler instance or 3rd party render application (i.e. Autodesk Maya render) on the burst render Amazon EC2 instances. Further customization of the CloudFormation templates is required to provide segregated render scheduling from a single management AWS account and Amazon VPC with multiple per-production project AWS accounts (see #1.1) and Amazon VPCs and their burst render Amazon EC2 instances, if supported by the render scheduler application functionality.
This section (#10.1 to #10.11) specifically describes using Deadline as the license-free on AWS render scheduler, but in a way so that the recommendations are also translatable to other non-Deadline render schedulers. For the purpose of describing controls in this section that are applicable to both Deadline and also to other non-Deadline render schedulers, the Deadline render scheduler is described as consisting of the Deadline Database and the Deadline Repository file-share both running on the same render scheduler instance. This section (#10.1 to #10.11) describes the submission of render jobs submitted from Deadline Clients (in-cloud or on-premise) to run on the available burst render Amazon EC2 instances via Deadline render scheduling. We recommend the manual configuration of Deadline (rather than automated wizard installers). Deadline requires the render scheduler instance and the burst render Amazon EC2 instances to be in the same AWS account. We recommend:-

- As per #1.1 determine the Data Tiering Classification and whether the new production requires a separate OU Member account per-production project to ensure segregated resource.
    - Tier 1 high-security content will require a separate OU Member account per-production project.
    - Tier 2/Tier 3 content may not require a separate OU Member account per-production project.
- Using a separate OU Member account per-production project when combined with the Deadline render scheduler functionality forces limited combinations of the deployable, architectural combinations of AWS account, Amazon VPCs, render scheduler and burst render resource to achieve the required granularity. These combinations are:-
    - The CloudFormation template deployed environment of a single AWS account containing the Deadline render scheduler instance in its single, management Amazon

VPC and burst render Amazon EC2 instances in its single, per-production Amazon VPC.
    - A single AWS account containing the Deadline render scheduler instance in its single management Amazon VPC and multiple sets of burst render Amazon EC2 instances each in their own single, per-production Amazon VPC. Each per-production Amazon VPC contains its own project dependent burst render Amazon EC2 instances. Deadline functionality does not permit the concurrent running of multiple per-production projects in different per-production Amazon VPCs from a single render scheduler. Deadline can be configured to switch between non-concurrent per-production project Amazon VPCs in the same AWS account, but this requires switching between separate sets of Deadline Databases and Repository for each per-production project.
    - A single AWS account containing the Deadline render scheduler instance in its single management Amazon VPC interacting with multiple, concurrent per-production projects where each per-production project is in its own AWS account and own per-production Amazon VPC. Deadline functionality does not permit the concurrent running of multiple per-production project accounts/VPCs from a single render scheduler account/VPC. This does not achieve the required granularity.
    - A single AWS account per-production project where the Deadline render scheduler and burst render Amazon EC2 instances are in the same AWS account and Amazon VPC. i.e. a fully segregated environment per-production. Deadline functionality supports this full segregation granularity, but each project requires its own Deadline render scheduler deployment.
- Determine the administrative/cost overhead of using separate AWS accounts for each per-production project related the data tiering classification, the ability to run multiple, concurrent projects and how this aligns to the above options requiring single/multiple Deadline render schedulers. The only way to enforce fully segregated production assets requires a completely different account/VPC for each project. This may be impractical administratively and financially for many VFX Studios working on multiple projects at the same time. Compromises in the configuration of the Deadline render scheduler relative to the segregation and granularity of using different AWS accounts/VPCs for each production will need to be made.
- Use AWS CloudTrail Logs to audit account usage by reviewing the log files that contain a history of AWS API calls and related events for the account.
- As per #10.11, the Deadline logs should be pushed to a central AWS account for auditing and compliance.

10.2 AWS Deadline - Use a secure connection between Client and Deadline Repository

The CloudFormation template does not use Deadline in the deployed burst rendering environment. We recommend the manual configuration of Deadline (rather than automated wizard installers).

Unlike other centralised render schedulers, Deadline scheduling is decentralised and takes place on the instances/servers running the Deadline Slave application of the Deadline Client. The individual Deadline Slaves execute the job scheduling logic to dequeue tasks instead of using a central render scheduler dispatcher. The only centralized Deadline components are the Deadline Database and the Deadline Repository file-share which, for the purpose of description in this section (#10.1 to #10.11), can both run on the right sized render scheduler instance to provide high availability and fault tolerance.
The Deadline Client is a group of loosely coupled applications running on the burst render Amazon EC2-instances and on-premises servers. The Deadline Slave is one application within the Deadline Client.
The Deadline Repository is described as running on the render scheduler. It is a network file-share for the render scheduling plugins, scripts, logs required by the render scheduled jobs, and optionally for bespoke input files that are submitted for control of the render scheduling jobs. The Deadline Repository should not contain the 3rd party render application scene and shot information and image assets; these are referenced directly by the burst render Amazon EC2 instances on network available asset storage. In practice to ensure that the Deadline Repository file-share runs as fast as possible at scale and to achieve better file-share performance, the Deadline Repository file-share is located on the fastest tier of network available asset storage and not on a render scheduler instance with fast local storage.
To enable the Deadline Clients to communicate securely with the Deadline Repository file-share we recommend this is done via https with transport layer security TLS/SSL, to access ancillary items required for the render job. This https secure communication is via the Deadline Remote Connection Server (RCS) and NGINX load balancer. Recommendations are:-

- The RCS provides the web server front end to the Deadline render scheduler (Database and Repository file-share) and is fronted by a NGINX load balancer and cache server to improve performance and scaling. Multiple RCS can front the Deadline Repository file-share. A NGINX load balancer and cache server can front the multiple RCS. Calls from the Deadline Client to retrieve data, scripts and plugin files from the Deadline Repository file-share will be cached in the NGINX layer.
- Enable Deadline Client communication with the Deadline Repository to use https with transport layer security (TLS/SSL) and certificate to encrypt communications and authenticate Deadline Client connections. Deadline Client communication with the Deadline Repository is via the NGINX/RCS which handles the https TLS termination and load balancing. Access from on-premises Deadline Clients will be over the IPsec VPN (see #2.8) and AWS Direct Connect.
- Create least privilege policies that allows the Deadline Repository file-share running on the render scheduler instance in the management Amazon VPC to securely communicate with:-
    - The Deadline RCS(s) and NGINX fronting the Deadline Database and Repository file-share.
    - Each resource set (Amazon EC2 instances, Amazon EC2 Spot Fleet, asset storage, license server, logging, etc.) required to be operational.
- Create least privilege policies that allows the RCS and NGINX running on the render scheduler instance in the management VPC to securely communicate with:-
    - The Deadline Client running on the burst render Amazon EC2 instances in the per-production Amazon Virtual Private Cloud (Amazon VPCs). The Deadline Client

must have read access to the Repository file-share root and its subdirectories, read/write access to some subdirectories and read/execute to some subdirectories.
    - The Deadline Client running on on-premises named IP systems specifically the Launcher, Monitor and Submitter applications, over the IPsec VPN and AWS Direct Connect.
    - Note that TLS/SSL requires the creation of a password protected certificate that is installed on each Deadline Client, and the user running the Deadline Client requires read access to this certificate.
- Create the 'service principal' render scheduler AWS Identity and Access Management (IAM) role with the above least privilege policies. Use the 'service principal' render scheduler IAM role to run the Deadline render scheduler.
- Verify that the render scheduler IAM role activity is logged as CloudTrail events and that the necessary escalation alarm notification is in place for access discrepancies.

10.3 AWS Deadline - Use SSL based Authentication for Deadline Database

The CloudFormation template does not use Deadline in the deployed burst rendering environment. We recommend the manual configuration of Deadline (rather than automated wizard installers).
The Deadline Database is described as running on the render scheduler. It stores the render scheduled jobs, settings and configurations. The Deadline Database does not contain the 3rd party render application scene and shot information and image assets; these are referenced directly by the burst render Amazon EC2 instances on asset storage.
To enable the Deadline Clients to communicate securely with the Deadline Database we recommend this is done via https with transport layer security TLS/SSL, to access the ancillary items required for the render job. This https secure communication is via the Deadline Remote Connection Server (RCS) and NGINX load balancer. Recommendations are:-

- As per #10.2, enable Deadline Client communication with the Deadline Database to use https over transport layer security (TLS/SSL) and certificate to encrypt communications and authenticate Deadline Client connections. Deadline Client communication with the Deadline Database is via the NGINX/RCS which handles the https TLS termination and load balancing. Access from on-premises Deadline clients will be over the IPsec VPN (see #2.8) and AWS Direct Connect.
- Create least privilege policies that allows the Deadline Database running on the render scheduler instance in the management Amazon Virtual Private Cloud (Amazon VPC) to securely communicate with:-
    - The Deadline RCS(s) and NGINX fronting the Deadline Database.
    - Each resource set (Amazon EC2 instances, Amazon EC2 Spot Fleet, asset storage, license server, logging, etc.) required to be operational.
- Create least privilege policies that allows the RCS and NGINX running on the render scheduler instance in the management VPC to securely communicate with:-
    - The Deadline Client running on the burst render Amazon EC2 instances in the per-production Amazon VPC.

    - The Deadline Client running on on-premises named IP systems specifically the Launcher, Monitor and Submitter applications, over the IPsec VPN and AWS Direct Connect.
    - Note that TLS/SSL requires the creation of a password protected certificate that is installed on each Deadline Client, and the user running the Deadline Client requires read access to this certificate.
- Apply the Deadline Database privilege policies to the #10.2 'service principal' render scheduler AWS Identity and Access Management (IAM) role.
- Check that the Deadline Database has MongoDB authentication enabled in the /etc/mongod.conf file and use the MongoDB shell to validate it is currently running as enabled.

10.4 AWS Deadline - Use separate Deadline Repository based on client, workflow or workload

The CloudFormation template does not use Deadline in the deployed burst rendering environment. We recommend the manual configuration of Deadline (rather than automated wizard installers).
The Deadline Repository is described as running on the render scheduler. As per #10.2 it is a file-share for the render scheduling plugins, scripts, logs required by the render jobs, and optionally for bespoke input files that are submitted for control of the render jobs.
Deadline functionality does not currently permit the concurrent running of multiple, different per-production project Repository file-shares. Deadline can be configured to switch between non-concurrent per-production project Repository file-shares but this also requires the coordinated switching between non-concurrent per-production project Databases. The Repository and Database act together as a render scheduler set. Using a separate Deadline Repository for each client or workflow (hence per-production ‘air-gapped’ project) therefore requires using a separate Deadline Database for the per-production project. We recommend:-

- As per #10.7 the 'user security' settings of the Deadline Repository can be customized to define the group/user permissions of named users to access areas of the Deadline Repository. A single Deadline Repository (and Database) can therefore have granular user access permissions for different projects, but again as per #10.7 this aggressive segregation of the 'user security' settings can lead to Render Wrangler and user disruption.

- As per #10.1, the administrative/cost overhead of using separate Deadline Repository (and Database) needs to be considered relative to the per-production project related data tiering classification and the ability to run multiple, concurrent projects. This may be impractical administratively and financially for many VFX Studios working on multiple projects at the same time. Compromises in the configuration of the Deadline render scheduler relative to the segregation and granularity of using different AWS accounts/VPCs and different Repository/Database for each production will need to be made.

10.5 AWS Deadline - Create a Separate VPC and Security Group for Deadline's AWS Resources

The CloudFormation template deploys an example, generic render scheduler instance in the management VPC, and example burst render Amazon EC2 instances in the per-production Amazon Virtual Private Cloud (Amazon VPC), both within the same AWS account. This illustrates how a single render scheduler can fit into a segregated per-production project environment, assuming that the render scheduler functionality permits this level of segregation. The CloudFormation template uses Security Groups to control the inbound and outbound access between the render scheduler instance within the management Amazon VPC and the burst render Amazon EC2 instances in the per-production Amazon VPC, within the same AWS account.
We recommend further customization of the CloudFormation template provided example Security Groups to secure Deadline application limited traffic flow between the Deadline render scheduler instance within the management Amazon VPC and the Deadline Clients running on the burst render Amazon EC2 instances and on-premises servers. These are in addition to the normal OS ports needed for traditional services to operate correctly. We recommend:-

- As per #10.1 and #2.4 determine the Data Tiering Classification and whether the new production project requires a separate per-production project VPC to ensure segregated resource between projects.
    - Tier 1 high-security content will require a separate per-production project Amazon VPC that references the common management Amazon VPC.
    - Tier 2/Tier 3 content may not require a separate per-production project Amazon VPC. This may result in multiple different Tier2/Tier3 productions running in the same per-production Amazon VPC that reference the common management Amazon VPC.
- Create Security Groups (as per #2.5) to control inbound and outbound network traffic between the Deadline render scheduler (Database and Repository file-share) in the private subnets of the management Amazon VPC and:-
    - The Deadline Client running on the burst render Amazon EC2 instances in the per-production Amazon VPC.
    - The Deadline Client running on on-premises named IP systems specifically the Launcher, Monitor and Submitter applications, over the IPsec VPN and AWS Direct Connect.
    - Outbound access to the internet for cloud licenses for Usage Based Licensing (UBL) via the management Amazon VPC public front end subnet (via the NAT gateway and Internet Gateway).
- Supplement the use of Security Groups between instances by additionally defining the Network ACLs (as per #2.6) between the multi-tiered subnets and the Amazon VPC gateway devices and the Amazon VPC endpoints.
- Enable the necessary on-premises firewall rules to connect the on-premises Deadline Client to the Deadline render scheduler (Database and Repository file-share) in the management Amazon VPC, over the IPsec VPN and AWS Direct Connect.
- As per #10.1, the administrative/cost overhead of using segregated Deadline resource needs to be considered relative to the per-production project related data tiering classification and the ability to run multiple, concurrent projects. This may be impractical administratively and financially for many VFX Studios working on multiple projects at the same time.

Compromises in the configuration of the Deadline render scheduler relative to the segregation and granularity of using different AWS accounts/VPCs and different Repository/Database for each production will need to be made.

- As per #8.1 use AWS Config to validate the integrity of the Amazon VPC, subnet and Security Group configuration. Verify that Amazon VPC Flow Logs are monitored by Amazon CloudWatch Logs and that the necessary escalation alarm notification is in place to react to unexpected inbound/outbound traffic at the Amazon VPC and subnet level. Perform regular auditing of the Amazon VPC and Security Group configuration.

10.6 AWS Deadline - Use Server-side Encryption for all Asset Transfer (S3) Storage

The CloudFormation template does not use Deadline in the deployed burst rendering environment. We recommend the manual configuration of Deadline (rather than automated wizard installers).
The described assets for transfer are the image assets and the 3rd party render application scene and shot information referenced on asset storage by the Deadline Slave burst render Amazon EC2 instances. The described assets are not the render scheduling plugins, scripts and logs required by the render scheduling jobs that are stored on the Deadline Repository file-share.
We recommend the further customization of the burst rendering environment to configure the transfer of assets between the on-premises storage and the in-cloud, server side encrypted storage types described in #5.1 that are wider than just Amazon S3. We recommend:-

- Create an asset transfer and asset storage system and methodology for the transfer of assets between on-premises storage and the encrypted asset storage repository (Amazon S3) and the asset storage cache used directly by the Slave burst render Amazon EC2 instances
    - The manual customization of Deadline is necessary to align the asset storage types described in #5.1 and transfer methods with the chosen configuration of the Deadline render scheduler and granularity of using different AWS accounts/VPCs and different Repository/Database discussed previously.
- Enable the asset storage systems configured as the Slave persistent asset storage cache and the persistent asset storage repository to use AWS KMS managed keys (see #9.1) for server-side encryption of the asset files using the asset customer master key (CMK) for the per-production project. Use AWS KMS “Customer managed keys” rather than using “AWS managed keys”.
- The IAM role used to launch the burst render Amazon EC2 instance (see #10.8) requires policy privileges (see #9.2) for accessing AWS KMS and the per-production project CMK's to be able to generate encrypt/decrypt the data keys used on the asset storage.
- Create the secure asset transfer method for:-
    - Source assets:
        - Source asset transfer between the on-premises storage and the server-side encrypted, persistent asset storage repository (Amazon S3) in the per-production Amazon VPC over IPsec VPN and AWS Direct Connect.
        - Source asset transfer between the server-side encrypted, persistent asset storage repository (Amazon S3) and the server-side encrypted, persistent asset

storage cache used by the Deadline Slaves in the per-production Amazon VPC.
    - Rendered assets:
        - Rendered asset transfer between the server-side encrypted, persistent asset storage cache used by the Deadline Slaves and the server-side encrypted, persistent asset storage repository (Amazon S3) in the per-production Amazon VPC.
        - Rendered asset transfer between the server-side encrypted, persistent asset storage repository (Amazon S3) in the per-production Amazon VPC and the on-premises storage over IPsec VPN and AWS Direct Connect.
- Note that the Deadline automated wizard installer performs the above steps by creating the Asset Transfer System (ATS) for the secure transfer of assets between on-premises storage and the Deadline Slaves. ATS uses the Asset Transfer Server service/daemon to transfer assets between the on-premises storage and the persistent asset storage repository (Amazon S3) via a secure SSH tunnel over https. The Central Controller daemon communicates with the Asset Transfer Server and Slave Controller daemon (running on burst render Amazon EC2 instances) to coordinate asset transfers between the on-premises storage and the persistent asset storage repository. The Slave Controller daemon communicates with the Central Controller daemon to copy data between the persistent asset storage repository and the persistent asset storage cache used directly by the Deadline Slaves. The asset storage is not server-side encrypted by default and this server-side encryption needs to be enabled once the Amazon S3 bucket has been created.
- It may be practical for the persistent asset storage repository (Amazon S3) to be bypassed so that the server-side encrypted, persistent asset storage cache used by the Deadline Slave syncs directly with the on-premises storage.

10.7 AWS Deadline - Specify a set of Client machines as administrator for AWS

The CloudFormation template does not use Deadline in the deployed burst rendering environment. We recommend the manual configuration of Deadline (rather than automated wizard installers).
We recommend the further customization of the burst rendering environment to align the administration methods for the render scheduler instance and the Deadline render scheduler application relative to the granularity of using different AWS accounts/VPCs and different Repository/Database discussed previously. We recommend:-

- Administrative access to the Deadline render scheduler instance and the burst render Amazon EC2 instances is via the #4.1 AWS Systems Manager Session Manager without requiring to open inbound SSH ports. Access should be restricted to a limited set of source IPs so that only known administration machines can access the instances.
- Administrative access to the Deadline render scheduler application is via the Deadline Monitor. A machine running the Deadline Client can run the Deadline Monitor and Deadline Clients that communicate with the Deadline render scheduler. Administrative access from the Deadline Monitor is controlled by user permissions rather than source IP permissions.
    - The Deadline Monitor is a Deadline Client application launched from the Deadline Launcher (as executable or service/daemon). It communicates securely with the

Deadline render scheduler (Database and Repository file-share) via https with transport layer security TLS/SSL via the Deadline Remote Connection Server (RCS) and NGINX load balancer.

- Deadline administrators should protect the Deadline Monitor superuser password. The Deadline Monitor enables the administrator to:-
  - Deadline Repository 'enhanced user security' should be enabled to enforce the render is run as the submitting user configured via the Deadline Monitor 'render job as user' setting.
  - Manage the 'user security' settings of the Deadline Repository (via the Configure Repository Options) to define the functionality available to groups of named users for group permission options and job access levels to be able to view and modify other users' jobs.
- Note that a render wrangler function using the Deadline Monitor and Submitter scripts may potentially need to review, submit and inspect multiple render jobs, from multiple per-production projects, from multiple per-production accounts and VPCs. Aggressive setting of the 'user security' settings can lead to disruption when certain users are unable to see the full list of render jobs that are hogging or hanging the burst render Amazon EC2 instances.

- Verify that the AWS Identity and Access Management (IAM) user and role activity is logged as AWS CloudTrail Events and that the necessary escalation alarm notification is in place for access discrepancies.

10.8 AWS Deadline - Create and Use a preconfigured client image for EC2 Instances

The CloudFormation template does not use Deadline in the deployed burst rendering environment. We recommend the manual configuration of Deadline (rather than automated wizard installers).
We recommend that the burst render Amazon EC2 instance image used for the Deadline Client can be either:-

- A Thinkbox owned, public AMI based on Amazon Linux that is already hardened and loaded with the Deadline Slave and the required 3rd party render application, and validated for use as-is by Thinkbox for use by Deadline.
- Or as per #4.2, create a customized reference burst rendering image from an AWS originated AMI. This customized AMI must be hardened and loaded with the Deadline Slave and the required 3rd party render application.

10.9 AWS Deadline - Remove Storage and Metadata Associated with Render Nodes

The CloudFormation template does not use Deadline in the deployed burst rendering environment. We recommend the manual configuration of Deadline (rather than automated wizard installers).

- Verify the instance storage location of temporary render job scheduling data copied from the on-premises Deadline render scheduler and temporary 3rd party render application data or checkpointed render output.
    - Instance store: The data in a burst render Amazon EC2 instance store (HDD, SSD, NVMe SSD) persists only during the lifetime of the instance. If the instance reboots the data in the instance store persists. However if the instance is terminated or stops the instance store is reset and the data in the instance store does not persist.
    - Amazon Elastic Block Store (Amazon EBS) root volume: By default the root volume of a burst render Amazon EC2 instance will be deleted when the instance is terminated.
    - Additionally attached Amazon EBS volumes: By default the data on additionally attached Amazon EBS volumes of a burst render Amazon EC2 instance will NOT be deleted when the instance is terminated. The data persists after the instance termination.
        - Ensure that the DeleteOnTermination attribute of an Amazon EBS volume used specifically for the storage of instance temporary data during the instance lifespan is set to true to delete the data on instance termination. Do NOT do this for the #10.6 persistent asset storage cache used to store asset data for use by the burst render Amazon EC2 instances. Use AWS Config to verify.
    - Note that if the instance is hibernated or stopped rather than terminated, the data on the encrypted Amazon EBS root volume and additionally attached encrypted Amazon EBS volumes will be persisted. Instance store-backed instances cannot be hibernated.
- Upon the completion of a per-production project, remove the render resource and storage that was created via the #1.6 method of deploying consistent per-production render environments for each per-production project. Run cleanup scripts to:-
    - Remove the source asset files and rendered asset files from the server-side encrypted, persistent asset storage cache storage systems used by the Deadline Slaves. These could be additionally attached Amazon EBS volumes running a 3rd party parallel file system, or could be Amazon EFS.
    - Remove the source asset files and rendered asset files from the server-side encrypted, persistent Amazon Simple Storage Service (Amazon S3) asset storage used to sync with the server-side encrypted, persistent asset storage cache storage systems.
    - Disable the use of the AWS Key Management Service (AWS KMS) managed asset customer master key (CMK) used for the server-side encryption of the asset files for the per-production project.

10.10 AWS Deadline - Encrypt and Validate all Remote Commands

The CloudFormation template does not use Deadline in the deployed burst rendering environment. We recommend the manual configuration of Deadline (rather than automated wizard installers).
Deadline commands are scripted in un-compiled languages such as Python and use custom scripts and custom plugins to customize various aspects of Deadline such as submitting jobs or automating specific tasks after a job completes. Python scripts or shell commands can be directly executed via the Deadline command line, or can be used from within the existing Deadline Monitor Script

environment (jobs, tasks, slaves, etc) or from within the existing Deadline Plugin environment (application, event, cloud, balance). We recommend:-

- Disable the ability of the Deadline Monitor to remotely administer remote Deadline Client machines.
    - Remote administration of Deadline Clients from the Deadline Monitor running on another machine is disabled by default. Use the Deadline Monitor (via the Configure Repository Options) to verify that the 'Client Setup', 'Remote Control' 'remote administration' setting is disabled. This disables the ability of the Deadline Monitor to remotely execute shell commands on Deadline Clients.
    - If and when remote administration of Deadline Clients is required from the Monitor, enable the whitelist of a limited set of specific commands that can be remotely executed on the Deadline Clients.
- The use of Deadline plugins (CommandLine, CommandScript) are disabled by default
    - Use the Deadline Monitor (via the Configure Repository Options) to verify that the 'Client Setup', 'General' 'applications plugins enabled by default' setting is disabled.
    - Restrict write access to the <custom> folder of the Deadline Repository file-share, used to store the Custom Scripts and Custom Plugins, so that only administrators have the ability to update Custom Scripts and Custom Plugins.
- An action defined in the Custom Scripts and Custom Plugins will be performed by the Deadline Slave running on the burst render Amazon EC2 instances. This Slave application runs as the #10.2 'service principal' render scheduler AWS Identity and Access Management (IAM) role used to launch the burst render instance and the role has limited permissions to only access the Deadline Repository and the asset storage.

10.11 AWS Deadline - Centrally log all Deadline and AWS Events

The CloudFormation template does not use Deadline in the deployed burst rendering environment. We recommend the manual configuration of Deadline (rather than automated wizard installers).

Deadline event logs from the Deadline render scheduler and Deadline Slave can be visualized via the Deadline Monitor and exported from the Deadline Monitor or via the Deadline Command CLI. We recommend:-

- Administrative access to the Deadline render scheduler application is via the Deadline Monitor. A machine running the Deadline Client can run the Deadline Monitor and Deadline Clients need to communicate with the Deadline render scheduler. Administrative access from the Deadline Monitor is controlled by user permissions rather than source IP permissions.
- Use the Deadline Monitor (via the Configure Repository Options) to verify that the 'Application Data' settings:-
    - The 'Application Logs' has set the cleanup frequency of the application logs and has set the logging verbosity of each Deadline application.
    - The 'History Entries' has the required number of Repository, Job and Slave history entries for the required level of logging before overwrite.

- It is not currently possible to redirect/pipe the Deadline logging to a remote location or into a separate Logging AWS account as this will stop Job, Task and Slave reports from being visible in the Deadline Monitor. Instead, logs can be extracted by:-
    - Install and configure CloudWatch agent on render node AMI to push the individual Deadline client logs to central AWS auditing account.
    - Create a separate log parsing service via a Deadline Event Plugin to regularly query the Deadline logging information via the Deadline Scripting API for access to the Job, Task and Slave reports and push this via AWS API to CloudWatch in the central AWS auditing account.

Notices

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

The software included with this paper is licensed under the Apache License, Version 2.0 (the "License"). You may not use this file except in compliance with the License. A copy of the License is located at http://aws.amazon.com/apache2.0/ or in the "license" file accompanying this file. This code is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Document Revisions

Date	Change	In sections
April 2019	Initial publication	—