# Build a Secure Enterprise Machine Learning Platform on AWS

## AWS Technical Guide

# Build a Secure Enterprise Machine Learning Platform on AWS: AWS Technical Guide

# Table of Contents

# Build a Secure Enterprise Machine Learning Platform on AWS

Publication date: **May 11, 2021**

## Abstract

This whitepaper helps cloud engineers, security engineers, Machine Learning Ops (MLOps) engineers, and data scientists understand the various components of building a secure enterprise machine learning (ML) platform. It provides prescriptive guidance on building a secure ML platform on Amazon Web Services (AWS).

## Introduction

Building an enterprise ML platform for regulated industries such as financial services can be a complex architectural, operational, and governance challenge. There are many architecture design considerations, including AWS account design, networking architecture, security, automation pipelines, data management, and model serving architecture in an ML platform implementation. In addition, organizations need to think about operational considerations such as the monitoring of pipelines, model training, and production model hosting environment, as well as establishing incident response processes for the ML platform operation. Lastly, having strong governance controls such as guardrails, model management, auditability, and data and model lineage tracking are essential to meet the stringent regulatory and compliance requirements faced by regulated customers.

AWS provides a wide range of services for building highly flexible, secure, and scalable ML platforms for the most demanding use cases and requirements. This paper provides architecture patterns, code samples, and best practices for building an enterprise ML platform on AWS.

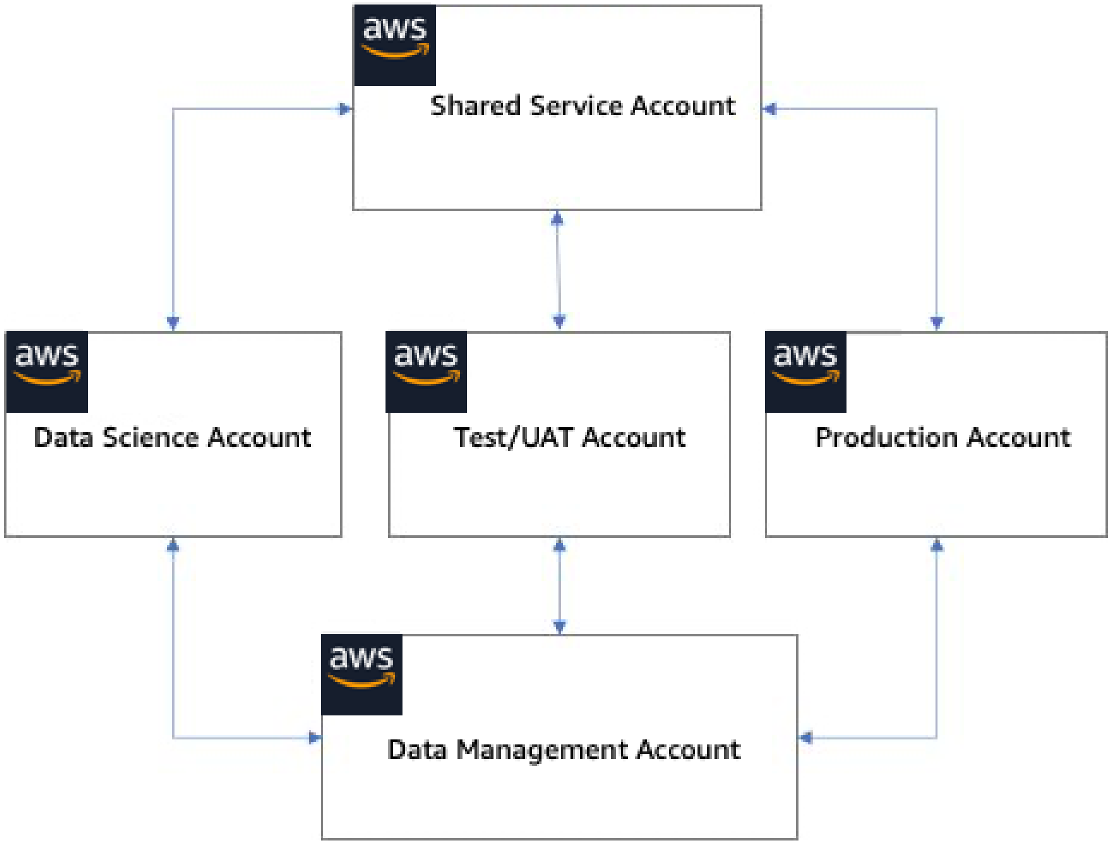
# Personas for an ML platform

Building an enterprise machine learning platform requires the collaboration of different cross-functional teams such as data scientists, cloud engineering, security architecture, data engineering, and audit and governance. The different personas from the different teams all contribute in the build-out, usage and operations of an ML platform, each having a different role and responsibilities. This document focuses on the following personas in an enterprise:

- **Cloud and security engineers** — In most organizations, cloud engineering and security engineering teams are responsible for creating, configuring, and managing the AWS accounts, and the resources in the accounts. They set up AWS accounts for the different lines of business and operating environments (for example, data science, user acceptance testing (UAT), production) and configure networking and security. Cloud and security engineers also work with other security functions, such as identity and access management, to set up the required users, roles, and policies to grant users and services permissions to perform various operations in the AWS accounts. On the governance front, cloud and security engineers implement governance controls such as resource tagging, audit trail, and other preventive and detective controls to meet both internal requirements and external regulations.
- **Data engineers** — Data engineers work closely with data scientists and ML engineers to help identify data sources, build out data management capabilities, and data processing pipelines. They establish security controls around data to enable both data science experimentation and automated pipelines. They are also responsible for data quality and data obfuscation management.
- **MLOps engineers** — MLOps engineers build and manage automation pipelines to operationalize the ML platform and ML pipelines for fully/partially automated CI/CD pipelines, such as pipelines for building Docker images, model training, and model deployment. They utilize different services such as pipeline tools, code repository, container repository, library package management, model management, and ML training and hosting platform to build and operate pipelines. MLOps engineers also have a role in overall platform governance such as data / model lineage, as well as infrastructure monitoring and model monitoring.
- **Data scientists and ML engineers** — Data scientists and ML engineers are the end-users of the platform. They use the platform for experimentation, such as exploratory data analysis, data preparation and feature engineering, model training and model validation. They also help analyze model monitoring results and determine if the model is performing as expected in production.
- **IT Auditors** — IT auditors are responsible for analyzing system access activities, identifying anomalies and violations, preparing audit reports for audit findings, and recommending remediations.
- **Model risk managers** — Model risk managers are responsible for ensuring machine learning models meet various external and internal control requirements such as model inventory, model explainability, model performance monitoring, and model lifecycle management.

# AWS accounts

Building an ML platform on AWS starts with setting up AWS accounts, and it is recommended to set up a multi-accounts architecture to meet the needs of an enterprise and its busines units. The following section discusses one multi-account pattern for building out an enterprise ML platform.



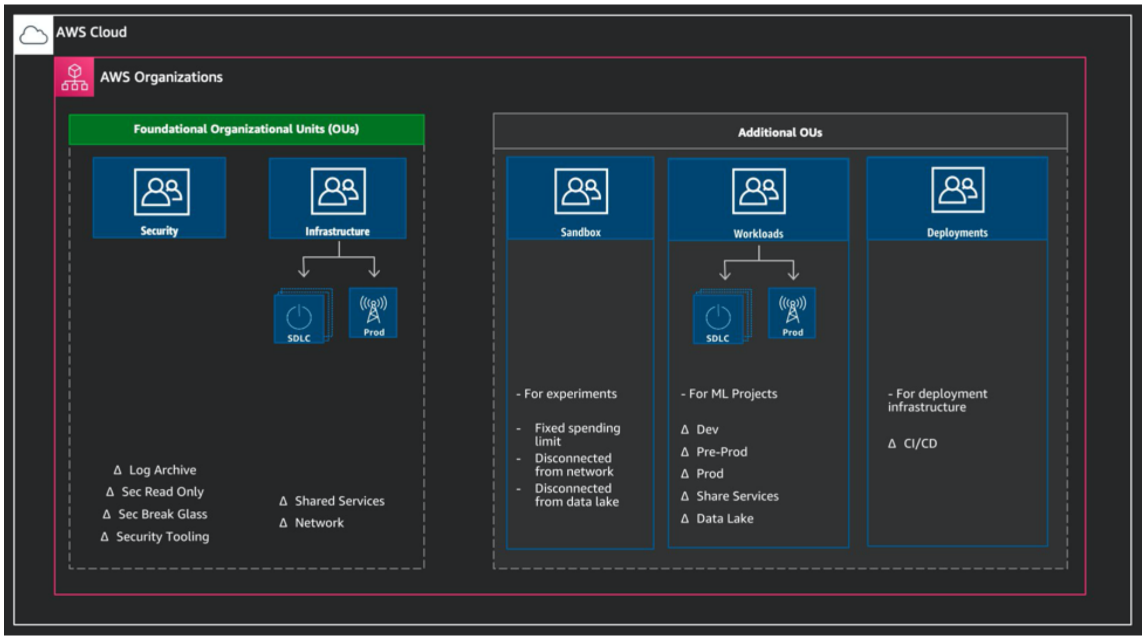
*AWS account design*

- **Shared Services account** — A Shared Services account is used to deploy and operate common services and resources within an enterprise ML platform. Common resources like shared code repositories, library package repositories, Docker image repositories, service catalog factory, and model repository can be hosted in the Shared Services Account. In addition to common resources, the Shared Services account would also host automation pipelines for end-to-end ML workflows. While it is not explicitly listed here, you also need to establish lower environments for the development and testing of common resources and services in the Shared Services account.
- **Data management account** — While data management is outside of this document's scope, it is recommended to have a separate data management AWS account that can feed data to the various machine learning workload or business unit accounts and is accessible from those accounts. Similar to the Shared Services account, data management also should have multiple environments for the development and testing of data services.
- **Data science account** — The data science account is used by data scientists and ML engineering to perform science experimentation. Data scientists use tools such as Amazon SageMaker to perform

exploratory data analysis against data sources such as Amazon Simple Storage Service (Amazon S3), and they build, train, and evaluate the model. They also have access to resources in the Shared Services account such as code, container, and library repositories, as well as access to on-prem resources. Note that data scientists need production datasets to build and train models in the data science account, so the access and distribution of data in the data science account need to be treated as data in a production environment.

- **Testing/UAT account** — MLOps engineers use testing/UAT accounts to build and train ML models in automated pipelines using automation services such as AWS CodePipeline and AWS CodeBuild hosted in the shared services account. Data scientists and ML engineers should not have change access to the testing/UAT account. If needed, they can be provided with read access to view model training metrics or training logs. If the model building and training workflows need to be tested in a lower environment before moving to the Testing/UAT account, the workflows can run in the data science account or a separate development account.
- **Production account** — The production account is used for production model deployment for both online inference and batch inference. Machine learning models should be deployed in the production account using automation pipelines defined in the Shared Services account.

You can use AWS Control Tower to build such a multi-account environment. You can determine what accounts are needed based on your requirements for account isolation requirements such business units or projects. AWS Control Tower offers you a mechanism to easily set up and govern a new, secure multi-account AWS environment. In AWS Control Tower, AWS Organizations helps centrally manage billing, access, control, compliance, security, and share resources across your member AWS accounts. Accounts are grouped into logical groups, called organizational units (OUs), as shown in the following figure.



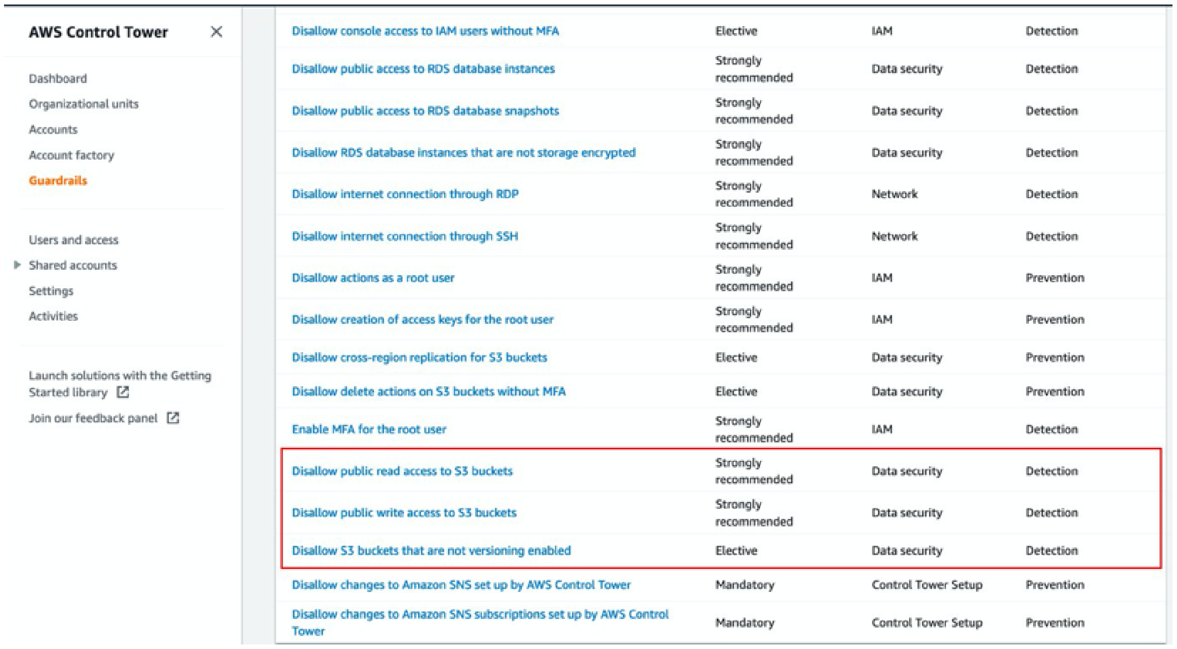
*AWS Organizations*

*AWS Organizations*

For machine learning projects, the accounts can be created within the Workloads OU or Line of Business (LoB) OU, as shown in the figure.

The creation of OUs enables you to set up organizational level pre-defined guardrails. These guardrails consist of AWS Config Rules that are designed to help you maintain the compliance of your environment. You can use them to identify and audit non-compliant resources that are launched in your environment; for example, for data protection, a common set of guardrails may be:

- Disallow public access to Amazon S3 buckets.
- Disallow S3 buckets that are not encrypted.
- Disallow S3 buckets that don't have versioning enabled.



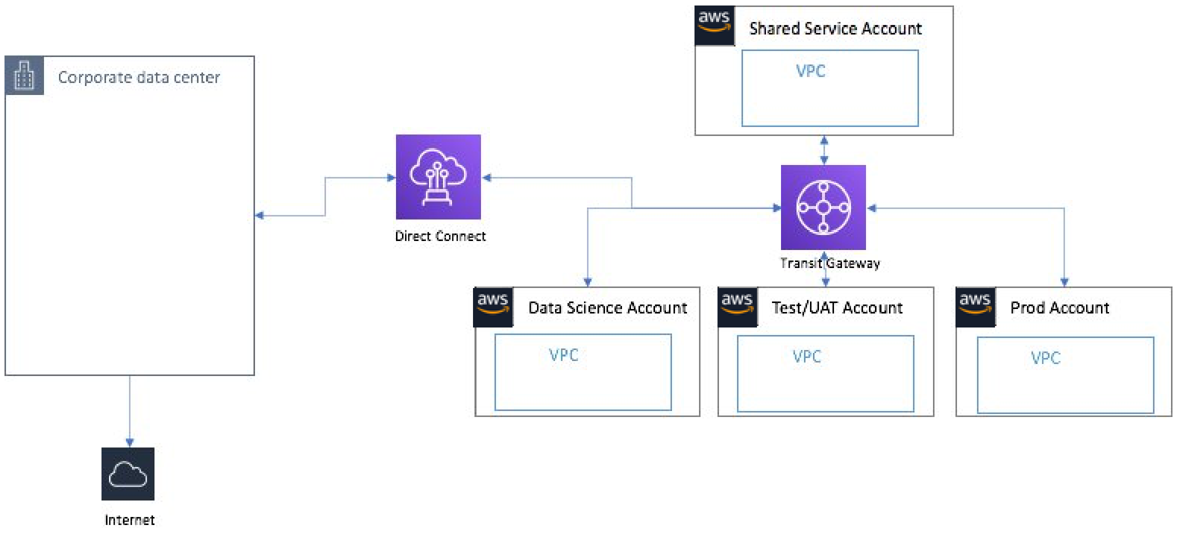
| | | | |
|---|---|---|---|
| Disallow console access to IAM users without MFA | Elective | IAM | Detection |
| Disallow public access to RDS database instances | Strongly recommended | Data security | Detection |
| Disallow public access to RDS database snapshots | Strongly recommended | Data security | Detection |
| Disallow RDS database instances that are not storage encrypted | Strongly recommended | Data security | Detection |
| Disallow internet connection through RDP | Strongly recommended | Network | Detection |
| Disallow internet connection through SSH | Strongly recommended | Network | Detection |
| Disallow actions as a root user | Strongly recommended | IAM | Prevention |
| Disallow creation of access keys for the root user | Strongly recommended | IAM | Prevention |
| Disallow cross-region replication for S3 buckets | Elective | Data security | Prevention |
| Disallow delete actions on S3 buckets without MFA | Elective | Data security | Prevention |
| Enable MFA for the root user | Strongly recommended | IAM | Detection |
| Disallow public read access to S3 buckets | Strongly recommended | Data security | Detection |
| Disallow public write access to S3 buckets | Strongly recommended | Data security | Detection |
| Disallow S3 buckets that are not versioning enabled | Elective | Data security | Detection |
| Disallow changes to Amazon SNS set up by AWS Control Tower | Mandatory | Control Tower Setup | Prevention |
| Disallow changes to Amazon SNS subscriptions set up by AWS Control Tower | Mandatory | Control Tower Setup | Prevention |

*AWS Control Tower Guardrails*

To set up additional guardrails, you can use Service Control Policies, which are described in more detail later in this document.

# Networking architecture

Enterprise ML platforms built on AWS normally have requirements to access on-premises resources, such as on-premises code repositories or databases. Secure communications such as AWS Direct Connect or VPN should be established. To enable flexible network routing across different AWS accounts and the on-prem network, consider using the AWS Transit Gateway service. If you want all internet traffic to go through your corporate network, configure an internet egress route to allow internet traffic to go through the on-premises network. The following figure shows a network design with multiple accounts and an on-premises environment.



*Networking design*

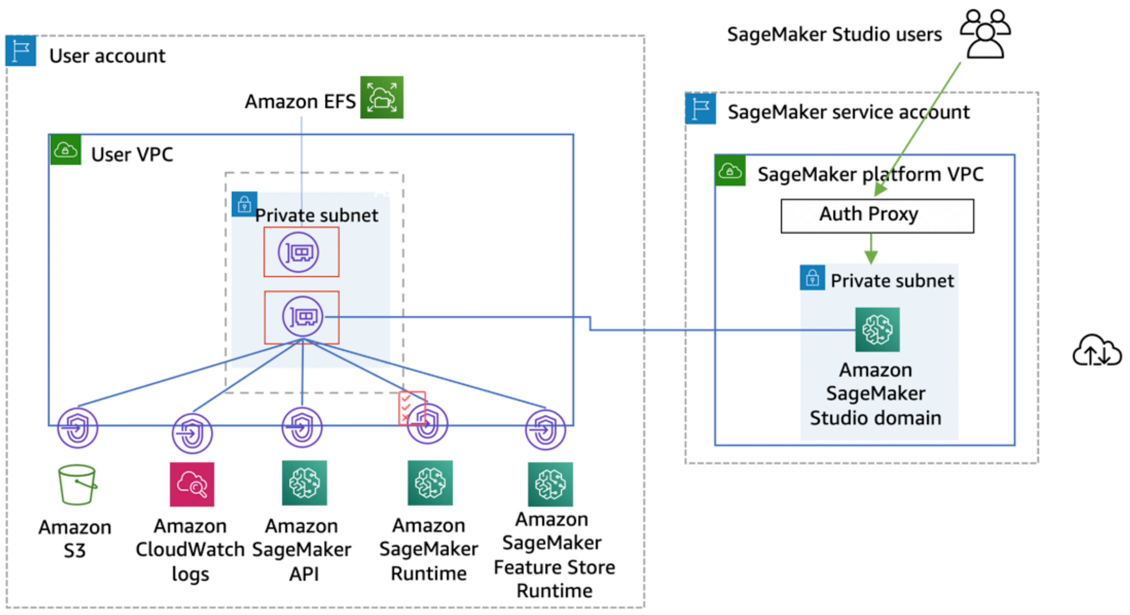
For enhanced network security, you can configure resources in different AWS accounts to communicate via the Amazon Virtual Private Cloud (VPC) using VPC endpoints. A VPC endpoint enables private connections between your VPC and supported AWS services. There are different types of VPC endpoints such as interface endpoint and gateway endpoint. An interface endpoint is an elastic network interface (ENI) with a private IP address from the IP address range of your subnet that you can control network access using a VPC security group. To access resources inside a VPC, you need to establish a route to the subnet where your interface endpoint is located. A gateway endpoint is a gateway that you specify as a target for a route in your route table. You can control access to resources behind a VPC endpoint using a VPC endpoint policy.

For data scientists to use Amazon SageMaker, AWS recommend the following VPC endpoints:

- Amazon S3
- Amazon SageMaker (to call SageMaker APIs)
- Amazon SageMaker Runtime (only use this in accounts which have permissions to invoke SageMaker endpoints)
- Amazon SageMaker Feature Store Runtime
- Amazon Security Token Service (STS)
- Amazon CloudWatch (for logging)
- AWS CloudTrail (for auditing API calls made by the service)
- Amazon Elastic Container Registry (ECR)
- AWS CodePipeline

- AWS CodeBuild
- AWS CodeArtifact

The following figure shows the networking architecture for SageMaker with private endpoints for all the dependent services.



*Networking architecture for Amazon SageMaker Studio inside a VPC (Not all VPC endpoints are shown for simplicity)*

# Identity and access management

To establish a secure ML environment, both human and machine identities need to be defined and created to allow for the intended access into the environment. AWS Identity and Access Management (IAM) is the primary access control mechanism you'll use for provisioning access to AWS resources in your environment. The IAM service provides capabilities for both identity management, such as support for identity federation, and authorization management for performing actions against AWS services such as Amazon SageMaker and Amazon S3.

For managing human users, identity federation is the recommended practice to allow for employee lifecycle events to be reflected in your ML environment when they are made in the source identity provider. You can set up identity federation to AWS using either AWS Single Sign-On (SSO) or IAM while leveraging your existing identity providers such as Okta or PingFederate. You can manage users and access to Amazon SageMaker Studio directly with AWS SSO, which enables user to sign in to SageMaker Studio with their existing corporate credentials.

After configuring identity management for your environment, you'll need to define the permissions necessary for your users and services. Following is a list of user and service roles to consider for your ML environment:

## User roles

User roles are assigned to the actual people who perform operations in an AWS account through AWS Management Console, The AWS Command Line Interface (AWS CLI), or APIs. Following are some of the primary user roles:

- **Data scientist/ML engineering role** — The IAM role for the data scientist / ML engineer persona provides access to the resources and services that are mainly used for experimentation. These services could include Amazon SageMaker Studio or SageMaker Notebook for data science notebook authoring, Amazon S3 for data access, and Amazon Athena for data querying against the data sources. Multiple such roles might be needed for the different data scientists or different teams of data scientists to ensure proper separation of data and resources.
- **Data engineering role** — The IAM role for the data engineering persona provides access to the resources and services mainly used for data management, data pipeline development, and operations. These services could include S3, AWS Glue, Amazon EMR, Athena, Amazon Relational Database Service (Amazon RDS), SageMaker Feature Store, and SageMaker Notebooks. Multiple such roles might be needed for the different data engineering teams to ensure proper separation of data and resources.
- **MLOps engineering role** — The IAM role for the MLOps persona provides access to the resources and services mainly used for building automation pipelines and infrastructure monitoring. These services could include SageMaker for model training and deployment, and services for ML workflow such as AWS CodePipeline, AWS CodeBuild, AWS CloudFormation, Amazon ECR, AWS Lambda, and AWS Step Functions.

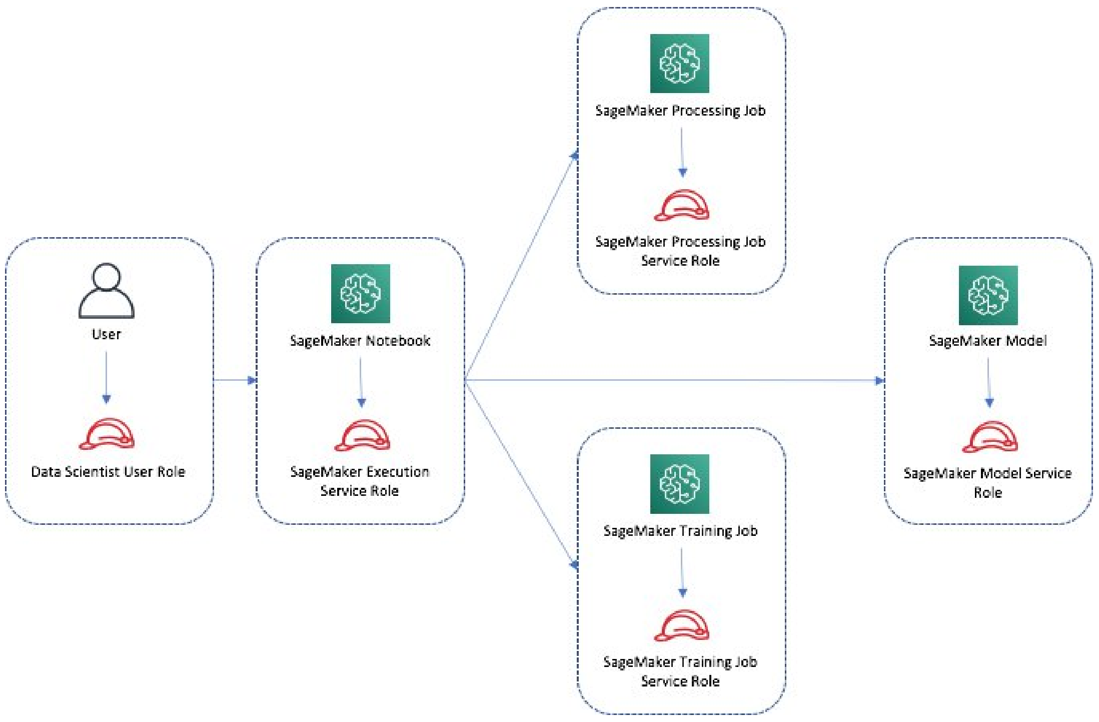
## Service roles

Services roles are assumed by AWS services to perform different tasks such as running a SageMaker training job or Step Functions workflow. Following are some of the main service roles:

- **SageMaker notebook execution role** — This role is assumed by a SageMaker Notebook instance or SageMaker Studio Application when code or AWS commands (such as CLI) are run in the notebook instance or Studio environment. This role provides access to resources such as the SageMaker training service, or hosting service from the notebook and Studio environment. This role is different from the data scientist / ML engineer user role.

- **SageMaker processing job role** — This role is assumed by SageMaker processing when a processing job is run. This role provides access to resources such as an S3 bucket to use for input and output for the job. While it might be feasible to use theSageMaker notebook executionrole to run the processing job, it is best practice to have this as a separate role to ensure it is in accordance with least privilege standard.
- **SageMaker training/tuning job role** — This role is assumed by the SageMaker training/tuning job when the job is run. Similarly, the SageMaker notebook executionrole can be used to run the training job. However, it is a good practice to have a separate role, which prevents giving end-users more rights than required.
- **SageMaker model execution role** — This role is assumed by the inference container hosting the model when deployed to a SageMaker endpoint or used by the SageMaker Batch Transform job.
- **Other service roles** — Other services such as AWS Glue, Step Functions, and CodePipeline also need service roles to assume when running a job or a pipeline.

The following figure shows the typical user and service roles for a SageMaker user and SageMaker service functions.


User
Data Scientist User Role
SageMaker Notebook
SageMaker Execution Service Role
SageMaker Processing Job
SageMaker Processing Job Service Role
SageMaker Training Job
SageMaker Training Job Service Role
SageMaker Model
SageMaker Model Service Role

*User role and services roles for Sagemaker*

## Permissions

IAM policies need to be created and attached to different roles to perform different operations. IAM provides fine-grained controls to allow / deny access to different SageMaker operations such as launching SageMaker Notebook instances or starting SageMaker training jobs. Following are some example IAM policies for controlling access to various SageMaker operations for the different roles. Note that the following IAM policies serve as examples only. It is important that you modify and test them for your specific needs.

- **Data scientist/ML engineer role** — Data scientists/ML engineers mainly need access to SageMaker Notebook instances or Studio for experimentation, or SageMaker console to view job status or other metadata. The following sample policies provide the data scientist / ML engineer role with controlled access to the SageMaker Notebook instance or SageMaker Studio domain.
- **SageMaker Console access** — The following sample policy enables an AWS user to gain read-only permission to the SageMaker console, so the user can navigate inside the console and perform additional privileged operations such as launching a SageMaker Notebook instance if additional permissions are granted in other policies. If you need to restrict read-only access to a subset of actions, you can replace `List*`, `Describe*`, and `Get*` with specific actions instead.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerReadAccess",
      "Effect": "Allow",
      "Action": [
        "sagemaker:List*",
        "sagemaker:Describe*",
        "sagemaker:Get*"
      ],
      "Resource": "*"
    }
  ]
}
```

- **SageMaker Notebook Access** — The following sample policy enables an AWS user to launch a SageMaker Notebook instance from the SageMaker console when the user has an AWS `userid` (for example, `AXXXXXXXXXXXXXXXXXXXX` or `<IAM Role ID>:<user name>` for a Security Assertion Markup Language (SAML) federated user) that matches the value of the “owner” tag associated with the notebook instance. The Governance section of this guide covers more detail on resource tagging and how it is used for permission management. The following IAM policy can be attached to an IAM user directly, or to an IAM role (for example, a data scientist role) that a user assumes.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerNotebookAccessbyOwner",
      "Effect": "Allow",
      "Action": [
        "sagemaker:StartNotebookInstance",
        "sagemaker:StopNotebookInstance",
        "sagemaker:CreatePresignedNotebookInstanceUrl"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "sagemaker:ResourceTag/owner": "${aws:userid}"
        }
      }
    }
  ]
}
```

The previous example uses `aws:userid` to manage fine-grained access to the SageMaker Notebook instances by the individual users. Another option is to use the `Session` tags and match the tag on the principal to resource, as shown in the following code sample. For more information about the `Principal` tag, see Working backward: From IAM policies and principal tags to standardized names and tags for your AWS resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerNotebookAccessbyOwner",
      "Effect": "Allow",
      "Action": [
        "sagemaker:StartNotebookInstance",
        "sagemaker:StopNotebookInstance",
        "sagemaker:CreatePresignedNotebookInstanceUrl"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "sagemaker:ResourceTag/owner": "${aws:PrincipalTag/owner}"
        }
      }
    }
  ]
}
```

- **SageMaker Studio access** — The following sample policy enables a SageMaker Studio user to access the SageMaker Studio where the user profile matches the user ID. This IAM policy can be attached to an IAM user directly, or an IAM role (for example, a data scientist role) that a user assumes. Similar to the previous example, you can also use `Session` tags and match the principal and resource tags in the condition. From an authentication perspective, SageMaker Studio also supports AWS Single-Sign-On based authentication.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerStudioAccessbyOwner"
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreatePresignedDomainUrl"
      ],
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "sagemaker:ResourceTag/owner": "${aws:userid}"
        }
      }
    }
  ]
```

- **SageMaker Notebook execution role** — The SageMaker notebook execution role needs access to data stored in S3, and permission to run SageMaker processing, training, or tuning jobs.

  The following sample policy allows a SageMaker notebook execution role to create a SageMaker processing, training, and tuning job and pass a job execution role to it.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerTraining",
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateTrainingJob",
        "sagemaker:CreateHyperParameterTuningJob",
        "sagemaker:CreateProcessingJob"
      ],
      "Resource": "*"
    },
    {
      "Sid": "SageMakerPassRoleTraining",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "<SAGEMAKER_TRAINING_EXECUTION_ROLE_ARN>",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "sagemaker.amazonaws.com"
        }
      }
    }
  ]
}
```

For quick experimentation, data scientists can build and push Docker images for model training to an Amazon ECR repo from the SageMaker Notebook instance. The following sample policy can be attached to the SageMaker Notebook execution role to enable this. The following policy also checks for ECR repos with resource tag equal to SageMaker to provide fine-grained access control to the different repos in the ECR. SageMaker also provides a suite of built-in algorithms containers and managed machine learning framework containers. These containers are accessible by various SageMaker jobs such as training jobs without the need for additional permission.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SagemakerCreateECR",
      "Effect": "Allow",
      "Action": [
        "ecr:CreateRepository"
      ],
      "Resource": "arn:aws:ecr:*:<ACCOUNT_ID>:repository/*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/CreatedBy": "SageMaker"
        }
      }
    },
    {
      "Sid": "SageMakerECRAccess",
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken"
      ],
      "Resource": "arn:aws:ecr:*:<ACCOUNT_ID>:repository/*"
    },
    {
      "Sid": "SagemakerECRRepo",
      "Effect": "Allow",
      "Action": [
        "ecr:BatchGetImage",
        "ecr:CompleteLayerUpload",
        "ecr:UploadLayerPart",
        "ecr:DescribeImages",
        "ecr:ListImages",
        "ecr:InitiateLayerUpload",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:PutImage"
      ],
      "Resource": "arn:aws:ecr:*:<ACCOUNT_ID>:repository/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/CreatedBy": "SageMaker"
        }
      }
    },
    {
      "Sid": "SagemakerECRRead",
      "Effect": "Allow",
      "Action": [
        "ecr:DescribeRepositories"
      ],
      "Resource": "arn:aws:ecr:*:*:repository/*"
    }
  ]
}
```

The following sample policy, when attached to the SageMaker notebook execution role, enables a user to create a model and deploy an endpoint in SageMaker.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerModel",
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateModel",
        "sagemaker:DescribeEndpointConfig",
        "sagemaker:CreateEndpointConfig",
        "sagemaker:CreateEndpoint",
        "sagemaker:DescribeEndpoint"
      ],
      "Resource": "*"
    },
    {
      "Sid": "SageMakerPassRoleModel",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "<SAGEMAKER_MODEL_EXECUTION_ROLE_ARN>",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "sagemaker.amazonaws.com"
        }
      }
    }
  ]
}
```

- **Training / tuning / processing job role** — When the SageMaker processing, training, or tuning job runs, it needs access to resources such as AWS Key Management Service (AWS KMS), CloudWatch Logs, and access to S3 data sources and ECR repository.

  The following sample shows a policy that can be attached to a training / tuning / processing job role to run the SageMaker training / processing / tuning job and use an S3 bucket as the input source and output target. This policy also allows the SageMaker job to create Elastic Network Interface (ENI) and communicate to other VPC resources with actions such as `ec2:CreateNetworkInterface`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerLog",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource": "*"
    },
    {
      "Sid": "SageMakerEC2Management",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:DeleteNetworkInterface",
        "ec2:DeleteNetworkInterfacePermission",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups"
      ],
      "Resource": "*"
    },
    {
      "Sid": "SageMakerKMSUsage",
      "Effect": "Allow",
      "Action": [
        "kms:CreateGrant",
        "kms:Decrypt",
        "kms:DescribeKey",
        "kms:Encrypt"
      ],
      "Resource": "<DATA_KMS_CMK_ARN>"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3::<ACCOUNT_ID>:<INPUT_BUCKET_NAME>",
        "arn:aws:s3::<ACCOUNT_ID>:<OUTPUT_BUCKET_NAME>/<PATH_NAME>"
      ]
    },
    {
      "Sid": "SageMakerECRAccess",
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken"
      ],
      "Resource": "arn:aws:ecr:*:<ACCOUNT_ID>:repository/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": "arn:aws:ecr:*:<ACCOUNT_ID>:repository/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/CreatedBy": "SageMaker"
        }
      }
    }
  ]
}
```

- **SageMaker Model hosting role** — the IAM policies for the SageMaker model will need access to EC2, AWS KMS, CloudWatch, and application auto-scaling to host the model in a SageMaker endpoint.

  The following example shows a policy that can be attached to the model hosting role to set up a SageMaker endpoint. You should further specify the resources to restrict access by the different actions based on requirements.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerLog",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource": "*"
    },
    {
      "Sid": "SageMakerEC2Management",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:DeleteNetworkInterface",
        "ec2:DeleteNetworkInterfacePermission",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups"
      ],
      "Resource": "*"
    },
    {
      "Sid": "SageMakerAutoscaling",
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:DeleteScalingPolicy",
        "application-autoscaling:DeleteScheduledAction",
        "application-autoscaling:DeregisterScalableTarget",
        "application-autoscaling:DescribeScalableTargets",
        "application-autoscaling:DescribeScalingActivities",
        "application-autoscaling:DescribeScalingPolicies",
        "application-autoscaling:DescribeScheduledActions",
        "application-autoscaling:PutScalingPolicy",
        "application-autoscaling:PutScheduledAction",
        "application-autoscaling:RegisterScalableTarget"
      ],
      "Resource": "*"
    },
    {
      "Sid": "SageMakerKMSUsage",
      "Effect": "Allow",
      "Action": [
        "kms:CreateGrant",
        "kms:Decrypt",
        "kms:DescribeKey",
        "kms:Encrypt"
      ],
      "Resource": "<DATA_KMS_CMK_ARN>"
    },
    {
      "Sid": "SageMakerECRAccess",
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken"
      ],
      "Resource": "arn:aws:ecr:*:<ACCOUNT_ID>:repository/*",
    },
    {
      "Sid": "SageMakerECRUsage",
      "Effect": "Allow",
      "Action": [
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": "arn:aws:ecr:*:<ACCOUNT_ID>:repository/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/CreatedBy": "SageMaker"
        }
      }
    },
    {
      "Sid": "SageMakerElasticInterface",
      "Effect": "Allow",
      "Action": [
        "elastic-inference:Connect"
      ],
      "Resource": "*"
    }
  ]
}
```

- **VPC endpoint** — You can create VPC endpoint policy to restrict access to resources behind VPC endpoints. The following policy will allow any user or service within the VPC to access the specified S3 buckets.

```
{
    "Version": "2012-10-17",
    "Sid": "AccessOnlytoSpecificBucket",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:ListAllMyBuckets"
            ],
            "Resource": [
                "arn:aws:s3:::<bucket_name>",
                "arn:aws:s3:::<bucket_name>/*"
            ]
        }
    ]
}
```

There are additional sample managed policies and custom policies that can be used as references for building IAM policies to meet different needs. For pushing containers from within Studio notebooks to ECR, see Using the Amazon SageMaker Studio Image Build CLI to build container images from your Studio notebooks. Guardrails can be set up at the account level to enforce policies such as network isolation or limiting training to a specific VPC. See the **Guardrails (p. 34)** section of this document for additional detail.

## Encryption with AWS KMS

Amazon SageMaker automatically encrypts model artifacts and storage volumes attached to training instances with AWS managed encryption key. All network traffic within the SageMaker service account and between the service account and your VPC is encrypted-in-transit using Transport Layer Security (TLS 1.2).

For regulated workloads with highly sensitive data, you might require data encryption using customer-managed Customer Master Key (CMK). The following set of AWS services provide data encryption support with CMK.

- SageMaker Processing, SageMaker Training (including AutoPilot), SageMaker Hosting (including Model Monitoring), SageMaker Batch Transform, SageMaker Notebook instance, SageMaker Feature Store, Amazon S3, AWS Glue, Amazon ECR, AWS CodeBuild, AWS Step Functions, AWS Lambda, Amazon EFS.

AWS KMS provides organizations with a fully managed service to centrally control their encryption keys. With AWS KMS, you can ensure your encryption keys are secure and available for the different services in the ML platform. For more information, see the *AWS Key Management Service Best Practices* whitepaper. If compliance needs dictate that keys must be frequently rotated, you can manually rotate the CMK with a new CMK. AWS KMS also rotates CMKs automatically once a year.

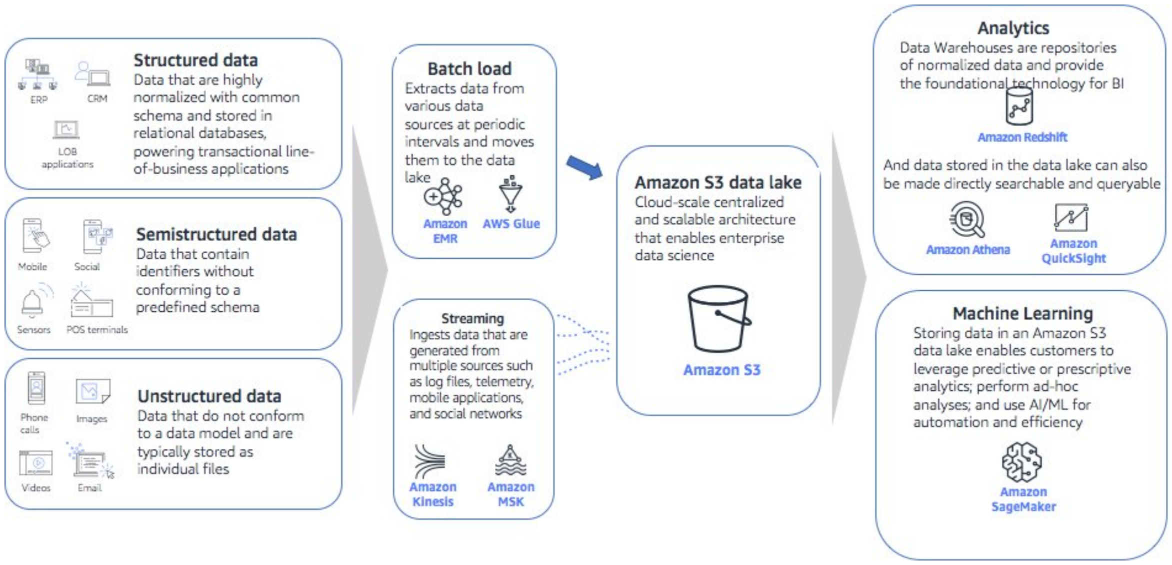
# Building the ML platform

To support data science workflows from experimentation to ML model productionization, an ML platform needs to provide several core platform capabilities, including:

- Data management with tooling for data ingestion, processing, distribution, orchestration, and data access control
- Data science experimentation environment with tooling for data analysis / preparation, model training / debugging / validation / deployment, access to code management repos and library packages, and self-service provisioning
- Workflow automation and CI/CD pipelines for automated model training, code / Docker image packaging, and model deployment in the production environment

## Data management

Amazon S3 is the primary storage for building an ML platform on AWS. The datasets needed for machine learning workflow should be ingested into S3 for easy and secure access from other data processing and machine learning services. AWS recommends that you build a data lake on S3 to centrally manage data storage, ETL, data catalog, and security. You can consider the AWS Lake Formation service to build out the data lake architecture. The following figure shows a high-level data management architecture for machine learning.


Structured data
Data that are highly normalized with common schema and stored in relational databases, powering transactional line-of-business applications
ERP
CRM
LOB applications
Semistructured data
Data that contain identifiers without conforming to a predefined schema
Mobile
Social
Sensors
POS terminals
Unstructured data
Data that do not conform to a data model and are typically stored as individual files
Phone calls
Images
Videos
Email
Batch load
Extracts data from various data sources at periodic intervals and moves them to the data lake
Amazon EMR
AWS Glue
Streaming
Ingests data that are generated from multiple sources such as log files, telemetry, mobile applications, and social networks
Amazon Kinesis
Amazon MSK
Amazon S3 data lake
Cloud-scale centralized and scalable architecture that enables enterprise data science
Amazon S3
Analytics
Data Warehouses are repositories of normalized data and provide the foundational technology for BI
Amazon Redshift
And data stored in the data lake can also be made directly searchable and queryable
Amazon Athena
Amazon QuickSight
Machine Learning
Storing data in an Amazon S3 data lake enables customers to leverage predictive or prescriptive analytics; perform ad-hoc analyses; and use AI/ML for automation and efficiency
Amazon SageMaker

*Data management architecture for analytics and machine learning*

Detailed enterprise data management design is out of scope for this document. The following section focuses on the S3 bucket design to support machine learning tasks in various environments.

Machine learning services such as SageMaker use Amazon S3 as the primary storage for experimentation, data processing, model training, and model hosting. To support data scientists' workflow and the automated CI/CD ML pipelines, a set of S3 buckets or folders should be created with proper data classification and data access control in mind. S3 provides security and access management features through IAM to control access to different buckets and folders. Follow known S3 best practices and other techniques such as data partitioning to optimize S3 performance. For machine learning workflow support, the following S3 buckets should be considered:

- **Buckets for users / project teams** — These buckets are used by individual data scientists or teams of data scientists with common access permission to the experimentation data. Data scientists pull data from the enterprise data management platform or other data sources and store them in these buckets for further wrangling and analysis. Data scientists also store training / validation / test datasets, and other experimentation artifacts such as training scripts and Docker files in these buckets for individual / team experimentations.
- **Buckets for common features** — Common features (for example, customer features, product features, and embedding features) that can be shared across different teams or projects can be stored in a set of common feature buckets. Organizational data access policies should govern access to these buckets, and consider providing individual data scientists with read-only access to these datasets if the data is not meant to be modified by individual data scientists. If you want to use a managed feature store, consider the SageMaker Feature Store to share features across teams.
- **Buckets for training / validation/test datasets** — For formal model training and tracking, the training / validation/test datasets for each model training pipeline need to be properly managed and versioned for traceability and reproducibility. These buckets will be the data sources for automated ML training pipelines.
- **Buckets for ML automation pipelines** — These are buckets used to store data needed for automation pipelines and should only be accessible by the automation services. For example, a bucket used by AWS CodePipeline to store artifacts (such as training script or Docker files) checked out from a code repository.
- **Buckets for models** — Model artifacts generated by the automation pipelines are stored in model buckets. Models used in production should be properly version controlled and tracked for lineage traceability.

# Data science experimentation environment

Data scientists and ML engineers use the experimentation environment for individual or team-based experimentations for the different science projects. The environment needs to provide services and tools for data querying and analysis, code authoring, data processing, model training and tuning, container development, model hosting testing, source code control, and data science library package access.

## Data science services

The following services can be provisioned in the experimentation environment in the data science account:

**Amazon Athena** — Data scientists and ML engineers can use Amazon Athena to query data in the data lake directly using SQL language. Athena provides an easy way to quickly identify and analyze a large amount of data directly against data stored in S3.

**Amazon SageMaker Notebook Instance and SageMaker Studio** — Data scientists and ML engineers can use a SageMaker Notebook instance or SageMaker Studio to author code, prepare data, and run model training directly inside a Jupyter environment. SageMaker Notebook instances and SageMaker Studio come pre-installed with a list of library packages and kernels (such as TensorFlow, Scikit-learn, and PyTorch) for easy model development.

If you use a SageMaker Notebook instance for code authoring and experimentation, the SageMaker Notebook instance provides Lifecycle script support, which can be used for:

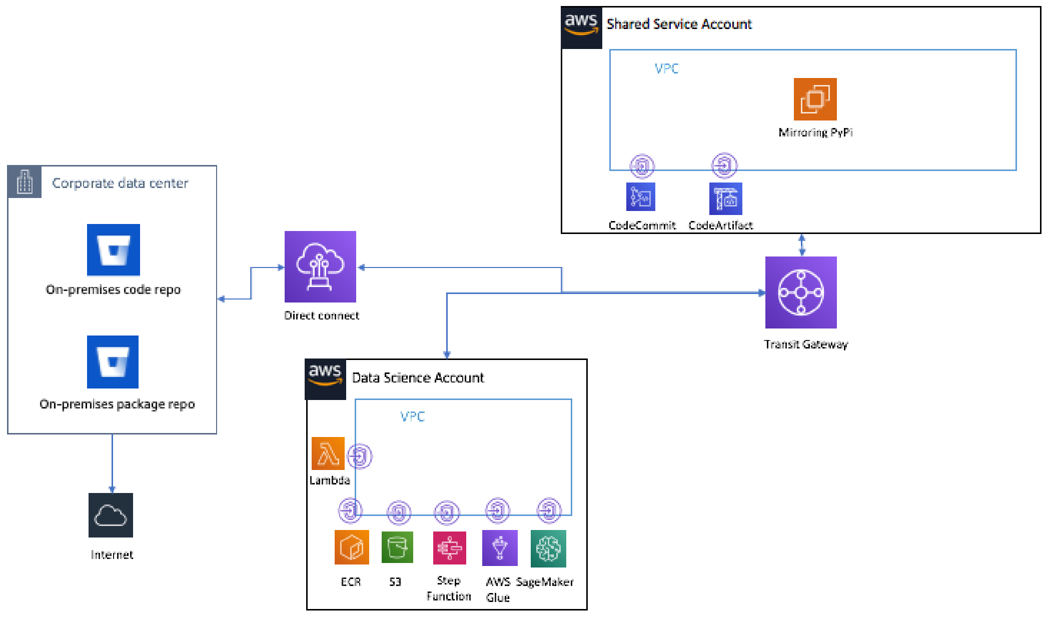
- Setting up environment variables such as VPC, Security Group, and KMS keys
- Configuring the Code Repo connection
- Configuring a connection to the internal package management server (such as ArtiFactory or CodeArtifact)

This sample CloudFormation script creates a Lifecycle configuration that sets up environment variables, configures a GitHub connection, and configure a PyPi mirror server. If you use SageMaker Studio, you can use the SageMaker Studio custom image to set up connection to private package management server. A SageMaker Studio custom image allows you to create a Docker image with your own selections of kernels, language packages, and other files to run a Jupyter notebook in SageMaker Studio. For more information, see Private package installation in Amazon SageMaker running in internet-free mode.

To use SageMaker Notebook instances or Studio in an enterprise environment, data scientists often need to provide infrastructure configuration information such as VPC configurations, KMS keys, and IAM roles for processing, training and hosing. To pass configurations to SageMaker training, processing jobs or model endpoints, consider using AWS Systems Manager Parameter Store to store these parameters in an encrypted fashion, and use a Python script to call these parameters via APIs. The Python script can be loaded onto the SageMaker notebook instance at startup using lifecycle configurations, or in a SageMaker Studio custom image.

- **Amazon SageMaker Data Wrangler (Data Wrangler)** — Data Wrangler is a feature of SageMaker Studio to import, transform, visualize, and analyze data. Data scientists can use Data Wrangler to perform data preparation tasks such as plotting histogram and scatter charts against datasets, running data transformations such as one-hot encoding, or handing data outliers.
- **Amazon SageMaker Processing** — Data scientists and ML engineers can use SageMaker processing for large data processing jobs. SageMaker processing provides built-in open-source containers for Scikit-learn and Spark. Data scientists can also bring custom containers to run processing jobs.
- **Amazon SageMaker Feature Store** — SageMaker Feature Store can help data scientists share common data features with other data scientists across teams for model training and inference. SageMaker Feature Store supports both offline feature store for training and online feature store for online inferencing.
- **Amazon SageMaker Training / Tuning service** — For model training and tuning, SageMaker provides fully managed model training and tuning services. It provides a list of built-in algorithms for different machine learning tasks such as classification, regression, clustering, computer vision, natural language processing, time series, and anomaly detection. It also provides a list of fully managed training open-source containers for TensorFlow, PyTorch, Apache MXNet, and Scikit-learn. Custom training containers can also be used for model training and tuning.
- **Amazon SageMaker Clarify (SageMaker Clarify)** — Data scientists and ML engineers can use SageMaker Clarify to compute pre-training and post-training bias metrics and feature attribution for explainability.
- **Amazon SageMaker Hosting** — Data scientists and ML engineers can test model deployment and real-time inference using the SageMaker hosting service. Models trained using the SageMaker built-in algorithms and managed containers can be deployed quickly using a single API command. Custom model inference containers can also be brought in to host custom models.
- **Amazon SageMaker Pipelines** — SageMaker Pipelines is a fully managed CI/CD service for machine learning. It can be used to automate various steps of the ML workflow such as data processing/transformation, training and tuning, and model deployment.
- **AWS Step Functions** — AWS Step Functions is a fully managed workflow orchestration tool. It comes with a data science SDK that provides easy integration of SageMaker services such as processing, training, tuning, and hosting. Data scientists and ML engineers can use AWS Step Functions to build workflow pipelines to automate the different steps (such as data processing and model training) in the experimentation environments.
- **Code repository** — A code repository such as Bitbucket or CodeCommit should be provided to data scientists and ML engineers for code management and version control. The code repository can reside in the Shared Services account or on-premises, and it is accessible from the data science account.
- **Amazon ECR (ECR)** — ECR is used to store training, processing, and inference containers. Data scientists and ML engineers can use ECR in the data science account to manage custom containers for experimentation.
- **Artifacts repository** — Organizations with strict internet access control often do not allow its users to download and install library packages from public package repositories directly, such as the Python

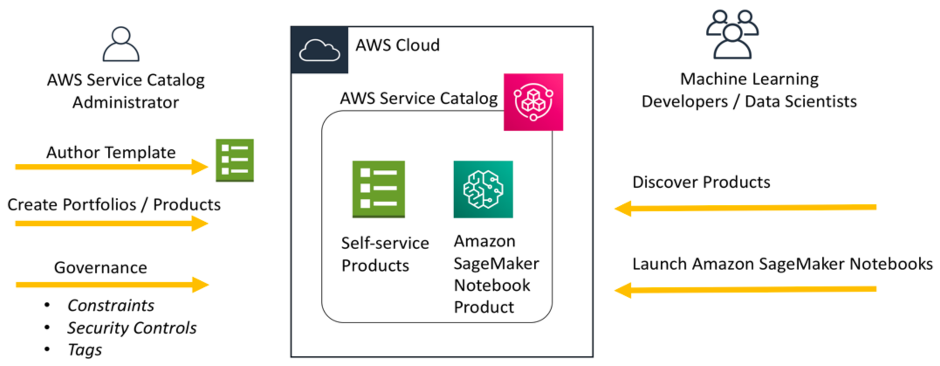
Package Index (PyPi) or Anaconda. Private package repositories such as Artifactory, AWS CodeArtifact, or mirroring PyPI servers can be created to support private packages management. These servers can be used to host private packages as well as a mirroring site for public package sites such as the PyPi for Pip or Anaconda main package channel and Conda-forge channel for Anaconda.



*Core components in the experimentation environment*

## Enabling self-service

To improve onboarding efficiency for data scientists and ML engineers, consider developing a self-service capability using the AWS Service Catalog. The AWS Service Catalog enables you to create self-service portfolio and products using CloudFormation scripts, and data scientists can directly request access to SageMaker Notebook / SageMaker Studio and other related AWS services without going through manual provisioning. Follow the instructions in Enable self-service, secured data science using Amazon SageMaker notebooks and AWS Service Catalog to enable self-service for data science products.


AWS Service Catalog Administrator
Author Template
Create Portfolios / Products
Governance
Constraints
Security Controls
Tags
AWS Cloud
AWS Service Catalog
Self-service Products
Amazon SageMaker Notebook Product
Machine Learning Developers / Data Scientists
Discover Products
Launch Amazon SageMaker Notebooks

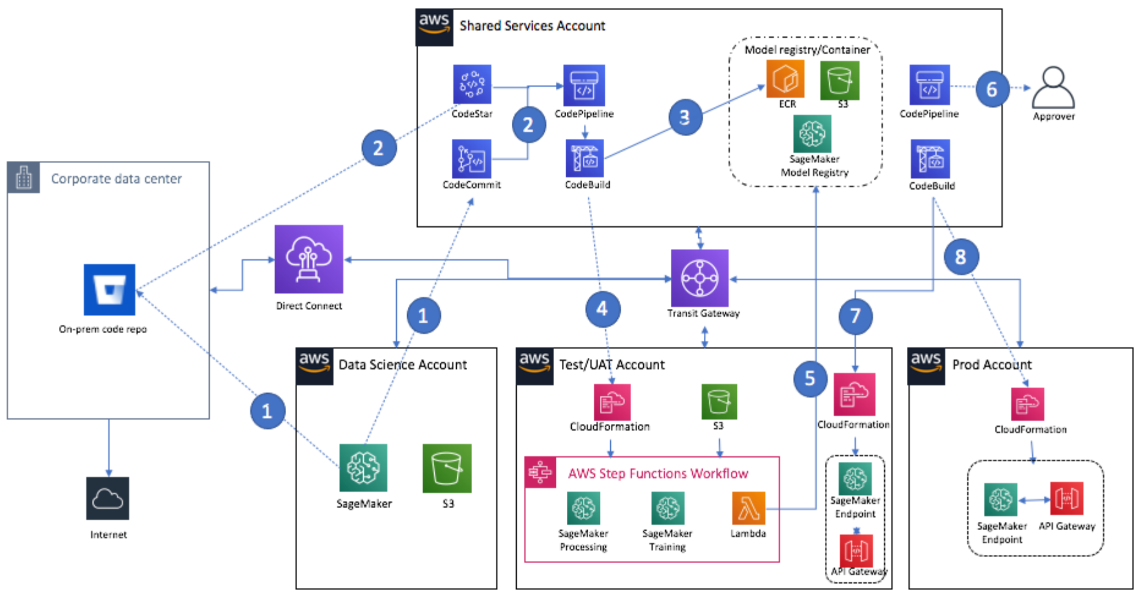
*Enabling self-service for data science products with AWS Service Catalog*

# Automation pipelines

Automated MLOps pipelines can enable formal and repeatable data processing, model training, model evaluation, and model deployment. In addition to process automation, MLOps pipelines can help enforce standards (for example, naming conventions, tagging, and security controls) and enable data and model lineage tracking. MLOps pipelines are built and operated by the MLOps engineering team, and they mainly consist of the following components:

- **Code repository** — A code repository is the source for an MLOps pipeline run. It contains artifacts such as docker files, training scripts, and other dependency packages for building containers, training models, and deploying models. Other artifacts such as AWS CloudFormation scripts and pipeline configuration templates are also stored in the code repository. A code repository can reside in the Shared Services account or in the on-prem environment.
- **Code build service** — A code build service is used to build custom artifacts such as custom Docker containers and push containers to a Docker image repository such as Amazon ECR. AWS CodeBuild or other third-party services such as Jenkins Build Server can be used as the code build service.
- **Data processing service** — Data processing service is used to process raw data into training/validation/testing datasets for model training purpose. SageMaker Processing can be used in an automated pipeline to process data.
- **Model training service** — SageMaker training service is used for model training in an automated pipeline. SageMaker training service takes input data sources from S3, train the model using a training container, and save the models in an S3 output bucket.
- **Model registry** — A model registry contains the metadata associated with model artifacts such as the location of the model artifacts, the associated inference container image, and the IAM role for running the container. Models can be registered in the Amazon SageMaker Model Registry for model inventory management and model deployment.
- **Model hosting service** — SageMaker provides a model hosting service for both real-time inference and batch inference. It takes model artifacts from the SageMaker Model Registry and deploys it to a serving endpoint.
- **Pipeline management** — The end-to-end pipeline management is controlled by the AWS CodePipeline service. AWS CodePipeline integrates with a code repository (such as AWS CodeCommit or BitBucket), and AWS CodeBuild. It also supports different deployment actions using AWS Lambda, AWS CloudFormation, and AWS Step Functions within the same account or across accounts. One of the main components in an end-to-end pipeline is the model build workflow orchestration for repeatable automated model building. The orchestration can be managed through services like SageMaker Pipelines or AWS Step Functions.

The following figure illustrates one MLOps pipeline reference architecture that works across multiple AWS accounts to build a custom container, process data, train a model, and deploy a model to an endpoint.



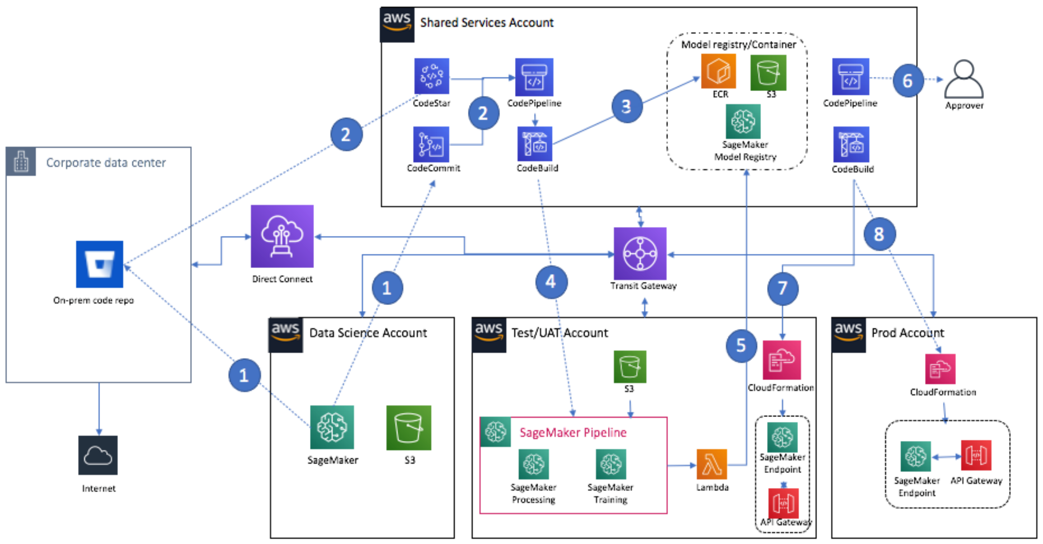
*Cross-account CI/CD flow for model training and deployment*

The pipeline in the figure consists of the following five stages:

1. **Change commit stage** — The data scientist/ML engineer commits code changes into a code repository (such as CodeCommit) from the data science account from the SageMaker notebook / Studio.
2. **Pipeline start stage** — A CodePipeline pipeline run is triggered in the Shared Services account. CodePipeline natively integrates with AWS CodeCommit, Enterprise Git, and BitBucket, downloads source codes from the code repository, and saves them into an S3 bucket used for CodePipeline inputs and outputs that's accessible by the different pipeline stages and action.
3. **Container build stage** — A CodeBuild project is then run as the next step in the pipeline in the Shared Services account. It uses the source code saved in the S3 bucket to build a custom Docker container and pushes the container image to the Amazon ECR repository in the Shared Services account.
4. **Model building stage** — Upon the successful Docker image build, the model training step is kicked off across the account (from the Shared Services account to the Test / UAT account) by launching a AWS CloudFormation script in the Test/UAT account. The AWS CloudFormation script first creates an AWS Step Functions state machine workflow consisting of a SageMaker processing step, a SageMaker model training step, and an endpoint deployment step in the test / UAT account for testing purposes. After the state machine is created, the CodePipeline triggers the Step Functions state machine to run all the steps defined in the state machine. The processing step and training step use containers hosted

in the Shared Services account Amazon ECR or public containers (built-in algorithms or managed containers) hosted by the SageMaker platform.

In addition to Step Functions, other workflow orchestration tools such as SageMaker Pipeline and AirFlow can also be used to orchestrate model training steps such as data processing, model training and model registration. SageMaker Pipeline enables you to create a workflow definition using the SageMaker SDK or a JSON pipeline definition. The CodeBuild job in the Shared Service account builds the SageMaker pipeline definition and runs the pipeline in the Test / UAT account by assuming an across-account role. You can also directly visualize the pipelines and track the running of differently pipelines directly inside SageMaker Studio.

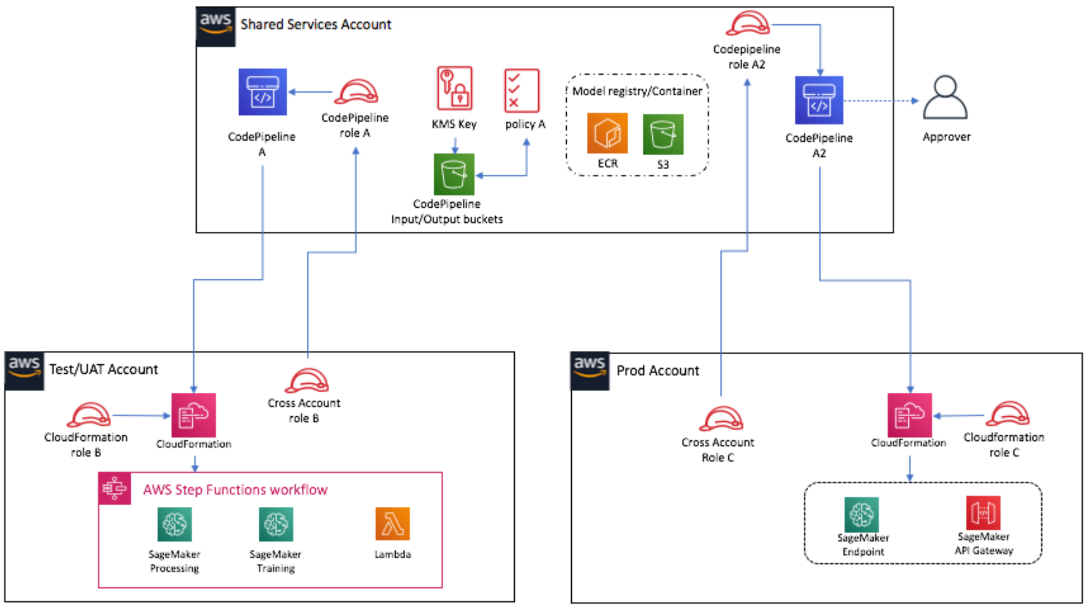


*CI/CD pipeline with SageMaker Pipeline*

5. **Model registration stage** — For centralized model management and inventory control. The trained model is registered in the SageMaker model registry in the Shared Services account. Any useful model metadata will also be saved in the central model registry for each model.
6. **Production deploymentstage** — When it is ready for production deployment, a separate CodePipeline workflow can be invoked in the Shared Services account to obtain release approval and run a CloudFormation script in the Production account to stand up a SageMaker endpoint for real-time inference. The production pipeline can be integrated with a release management ticketing system such as ServiceNow to raise a service ticket for tracking.

## Cross-account CodePipeline setup

When you create a pipeline with actions from multiple accounts, you must configure your actions with proper permission so that they can access resources within the limitations of cross-account pipelines. The following section reviews some of the key steps and components needed to enable a cross-account pipeline.



*Cross account pipeline access*

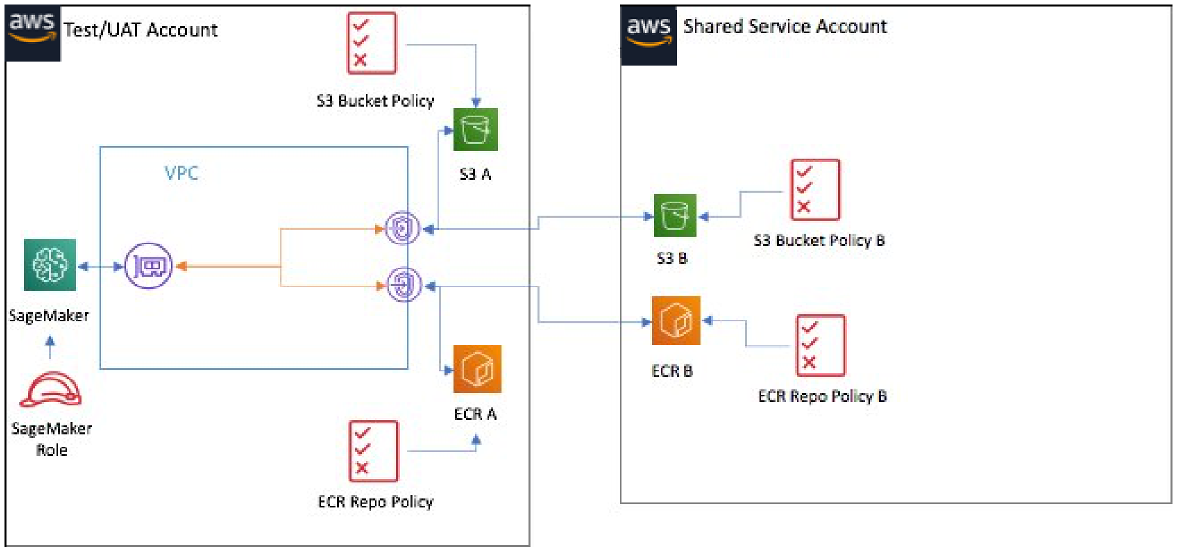
In the preceding figure, **CodePipeline A** represents the training pipeline, and **CodePipeline A2** represents the production deployment pipeline.

- **CodePipeline Role A** — This role is assumed by CodePipeline A to run the model training pipeline in the Test / UAT account using CloudFormation. This role has access to the KMS key used for data encryption for CodePipeline A’s input / output bucket. It also has full access to the CodePipeline input / output bucket.
- **Cross Account Role B** — This role is assumed by CodePipeline Role A to perform operations in the Test / UAT account. Specifically, this role is used to run the CloudFormation script to set up the Step Functions state machine for the training workflow, and invoke the Step Functions state machine after it is created.
- **Cross Account Role A2** — This role is assumed by CodePipeline A2 to run the model deployment pipeline in the production account.
- **Cross Account Role C** — This role is assumed by CodePipeline Role A2 to perform operations in the Production account. Specifically, this role is used to run the CloudFormation script to set up the SageMaker endpoint and other related resources such as Lambda functions and API Gateway to front the SageMaker endpoint.

- **CodePipeline Input/output bucket** — This bucket is used for sharing artifacts across different pipeline stages and different accounts. AWS Lambda supports bucket policies that can directly be attached to a bucket. In the preceding figure, Policy A is an S3 bucket policy that can provide different accounts with direct bucket access.
- **KMS Key** — A KMS key is required to encrypt data in the CodePipeline input/output bucket. The KMS key needs to provide access to the Test / UAT account, Production account, and CodePipeline role A. The KMS key is in the same region as the CodePipeline and CodePipeline input / output bucket. CodePipeline Role A will need access to this KMS key.

## Cross-account resource access via VPC endpoint

To ensure all network communication takes place within the private network, private endpoints should be used for accessing resources within the same AWS account or across different AWS accounts.



*Access resources across account via VPC endpoint*

The preceding figure illustrates how Amazon S3 and ECR can be accessed across accounts using VPC endpoints. In this diagram, SageMaker is attached to a VPC in the Test / UAT account through an Elastic Network Interface (ENI), and the VPC endpoints for S3 and ECR are attached to the Test / UAT account. To provide SageMaker role access to S3 buckets and ECR repositories in both the Test / UAT account and Shared Service account, you need to attach bucket policies and ECR repository policies that permit the SageMaker role to access the target S3 buckets and ECR repositories. Note that the S3 endpoint only supports buckets in the same AWS Region, and the ECR endpoint also only supports repositories in the same AWS Region.

## Cross-account pipeline example

To demonstrate how to build a cross-account pipeline using CodePipeline, review a sample CodePipeline definition with the following 2 stages:

- **Source stage** — This is the first stage in a pipeline that connects to a AWS CodeCommit repository. AWS CodeCommit is used to store any scripts for model training and CloudFormation templates used in the pipeline. This stage runs in the shared services account.
- **Deploy training workflow stage** — The second stage is for training and testing models in the Test account. First use a CloudFormation script to create a Step Functions state machine consisting of data processing and a model training step. Later you will invoke the created state machines to train a model.

**Sample CodePipeline definition with cross account support:**

```
{
    "pipeline": {
        "name": "<pipeline name>",
        "roleArn": "arn:aws:iam::<SHARED_SERVICE_ACCOUNT>:role/service-role/
<codepipeline_role_name>",
        "artifactStore": {
            "type": "S3",
            "location": "<S3 bucket name>",
            "encryptionKey": {
                "id": "arn:aws:kms:<REGION>: <SHARED_SERVICE_ACCOUNT>:alias/<kms key
 name>",
                "type": "KMS"
            }
        },
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "provider": "CodeCommit",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "BranchName": "<code branch name>",
                            "PollForSourceChanges": "false",
                            "RepositoryName": "<code repository name>"
                        },
                        "outputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "inputArtifacts": [],
                        "region": "<region>",
                        "namespace": "SourceVariables"
                    }
                ]
            },
            {
                "name": "Deploy_Training_Workflow",
                "actions": [
                    {
                        "name": "deploy-cf-train-test-external",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "provider": "CloudFormation",
                            "version": "1"
                        },
                        "namespace":"train-stack",
                        "runOrder": 1,
                        "configuration": {
                            "ActionMode": "CREATE_UPDATE",
                            "OutputFileName": "<output file name>",
                            "RoleArn": "arn:aws:iam::TEST_ACCOUNT:role/
<cloudformation_role_name>",
                            "StackName": "<CFN stack name>",
                            "TemplatePath": "SourceArtifact::<CFN template file name>"
                        },
                        "outputArtifacts": [
                            {
                                "name": "deploy-cf-train-test-output"
                            }
                        ],
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "roleArn": "arn:aws:iam::<TEST_ACCOUNT>:role/<Cross Account Role
 Name>",
                        "region": "<REGION>"
                    },
                    {
                        "name": "invoke-Step-Function",
                        "actionTypeId": {
                            "category": "Invoke",
                            "owner": "AWS",
                            "provider": "StepFunctions",
                            "version": "1"
                        },
                        "runOrder": 2,
                        "configuration": {
                            "InputType": "Literal",
                            "StateMachineArn": "#{train-stack.StateMachineARN}"
                        },
                        "outputArtifacts": [],
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "roleArn": "arn:aws:iam::<TEST_ACCOUNT>:role/<Cross Account Role
 Name>",
                        "region": "<REGION>"
                    }
                ]
            }
        ],
        "version": 10
    }
}
```

**Sample CloudFormation template for creating a Step Functions state machine for model training:**

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "An example template for a Step Functions state machine.",
  "Resources": {
    "MyStateMachine": {
      "Type": "AWS::StepFunctions::StateMachine",
      "Properties": {
        "StateMachineName": "SM-train-Step_Function",
        "StateMachineType": "STANDARD",
        "DefinitionSubstitutions": {
          "TrainingImageURL": "<training image ECR URL>",
          "RoleArn_CF": "<Cloudformation execution Role>",
          "Train_data_S3Uri": "<Training dataset S3 URI>",
          "Model_S3OutputPath": "<Model output path>"
        },
        "DefinitionS3Location": {
          "Bucket": "<Shared Services account bucket name>",
          "Key": "<Step Functions state machine definition json file name>"
        },
        "RoleArn": "<Step Function execution role>"
      }
    }
  },
  "Outputs": {
    "StateMachineARN": {
      "Description": "ARN of state machine",
      "Value": {
        "Ref": "MyStateMachine"
      }
    }
  }
}
```

## Deployment scenarios

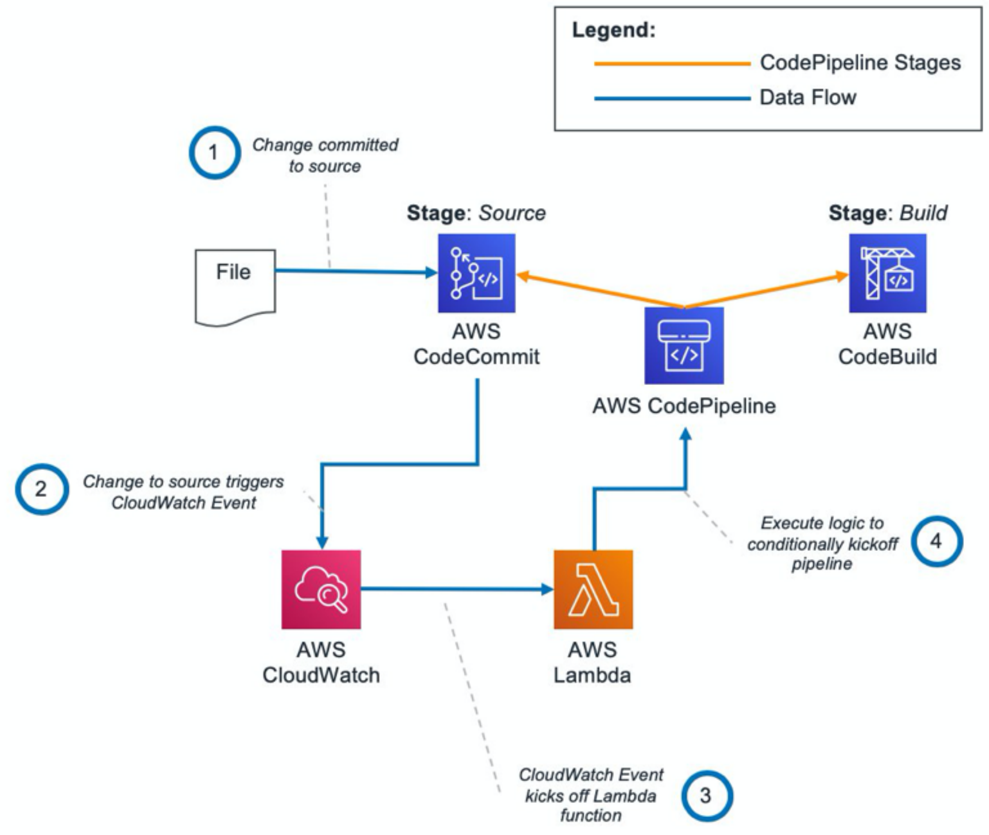
Depending on the deployment requirements, different pipelines need to be developed. Following are some examples of scenarios for deployment considerations:

- **End-to-end pipeline** — This pipeline includes stages such as the Docker image build, data processing, model training, approval, and model deployment. This pipeline builds everything from a source code repository and input data and deploys a model to a production endpoint.
- **Docker build pipeline** — This pipeline builds Docker images using source code (a Docker file) from the source code repository and pushes the images to a container registry such as Amazon ECR, along with additional metadata.
- **Model training pipeline** — This pipeline trains/retrains a model with an existing Docker container image and dataset, and optionally registers the model in the model registry after it is trained.
- **Model registration pipeline** — This pipeline registers an existing model saved in S3 and its associated inference container in ECR in the SageMaker model registry.
- **Model deployment pipeline** — This pipeline deploys an existing model from the SageMaker model registry to an endpoint.

There are several ways to trigger a pipeline run, including:

- A source code change
- A scheduled event
- On-demand via CLI / API

If source code change is used to trigger a pipeline run, you can either trigger the pipeline on any change in the source code repository, or only when a specific condition is met. If you use CodeCommit as the source, the following architecture pattern can be used to implement custom logic on whether to kick off a pipeline execution.


Legend:
CodePipeline Stages
Data Flow
1 Change committed to source
Stage: Source
Stage: Build
File
AWS CodeCommit
AWS CodePipeline
AWS CodeBuild
2 Change to source triggers CloudWatch Event
Execute logic to conditionally kickoff pipeline 4
AWS CloudWatch
AWS Lambda
CloudWatch Event kicks off Lambda function 3

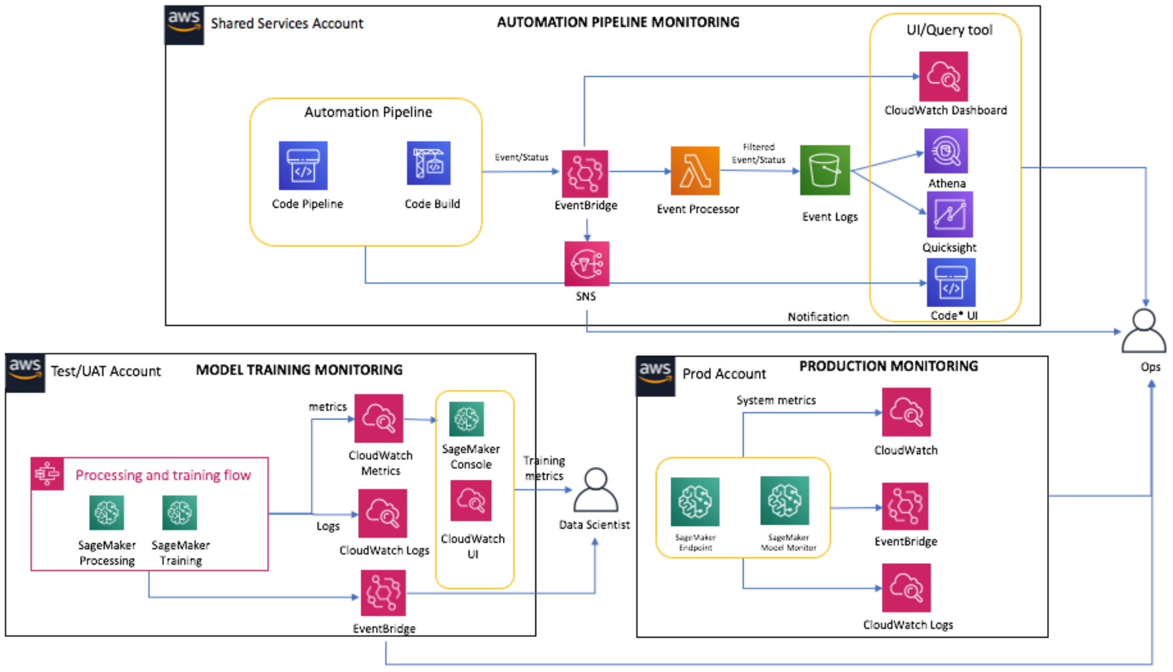
*Architecture pattern for kicking off pipeline based on custom logic*

# ML Platform Monitoring

ML Platform needs to be monitored for job status (such as training job failure/success), platform health (such as CPU / memory usage), and various metrics (such as inference error rates, data drift, and training loss). There are three main areas of an ML platform that require monitoring:

- Automation Pipeline
- Model training
- Production model serving

See the following figure for a sample monitoring architecture for the three monitoring areas:



*Sample ML platform monitoring architecture*

## Automation pipeline monitoring

*Sample ML platform monitoring architecture*

You can configure your AWS environment to monitor automation pipelines for pipeline status, and trigger notification when important events are detected. CodePipeline and CodeBuild provide event status on pipeline and build runs such as “in-progress” or “succeeded.” They also integrate with CloudWatch Events to log events to S3 for detailed analysis, or send out notifications via SNS service. For more information, see Monitoring pipelines  and Monitoring AWS CodeBuild.

## Model building monitoring

You can monitor the model building pipeline for training status and metric reporting. The various services used in the pipeline can provide job status and integrate with other AWS services such as

CloudWatch, EventBridge, and CloudTrail to support data collection, analysis, and real-time notification. For example, Step Functions can report status such as Success, Failure, Canceled, or In-Progress during its run. SageMaker emits status change events for SageMaker labeling, training, hyperparameter tuning, process jobs, and inference endpoint. It also sends training metrics for built-in algorithms such as mean absolute error (MAE) and accuracy, and custom-defined metrics for custom models to CloudWatch Metrics and training logs to CloudWatch Logs. If real-time notification is needed, you can use EventBridge to send out notification or trigger additional workflows. For more information, see Monitor Amazon SageMaker and Monitoring Step Step Functions Using CloudWatch.

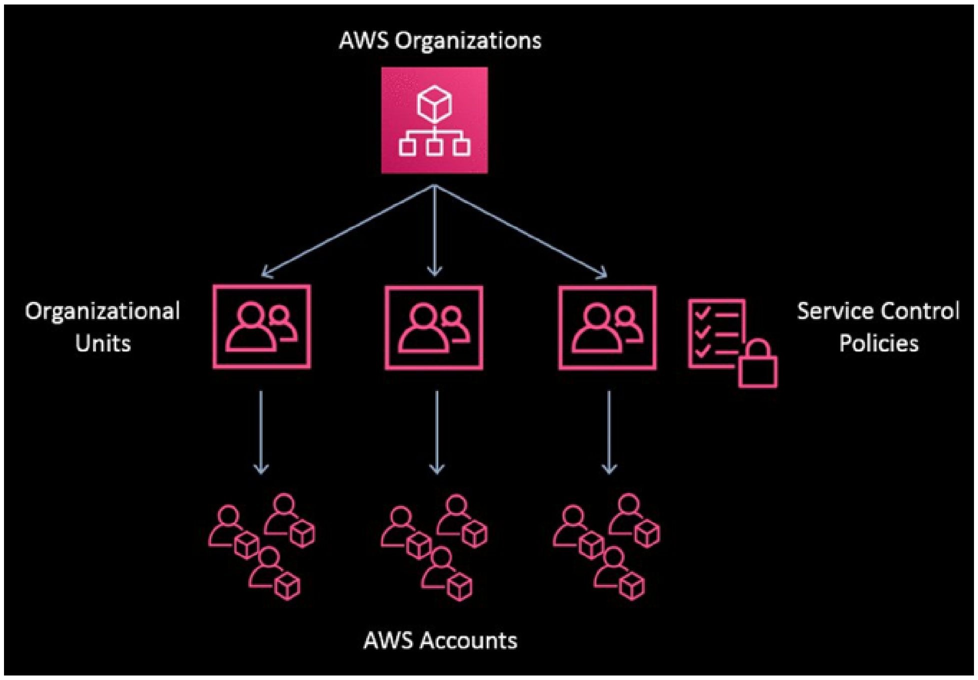
## Production endpoint monitoring

You can monitor production endpoints for system health, data drift, model drift, data and model bias, and explanation for each prediction. SageMaker endpoint reports a set of system metrics such as CPU and memory utilization, model invocation metrics, model latency, and errors through CloudWatch. SageMaker Model Monitor can detect data/concept drift of models hosted by the SageMaker hosting service. For model drift detection, model metrics can be periodically evaluated using collected labeled data running against the hosted model.

# Governance and control

## Guardrails

Large enterprises with strict security and compliance requirements need to set up guardrails for operating the ML environments. IAM policies can be used for enforcing guardrails, such as requiring proper resource tagging or limiting type of resources used, for different users and roles. For enterprise scale guardrail management, consider AWS Organizations. Its Service Control Policies (SCP) feature can help with enterprise guardrail management, by attaching a SCP to an AWS Organizations entity (root, organizational unit (OU), or account). You still need to attach identity-based or resource-based policies to IAM users or roles, or to the resources in your organization's accounts to actually grant permissions. When an IAM user or role belongs to an account that is a member of an organization, the SCPs can limit the user's or role's effective permissions.



*Managing guardrails with AWS Organizations and Service Control Policies*

## Enforcing encryption

- **Enforcing notebook encryption** — SageMaker Notebook Instance EBS volume encryption can be enforced using the `sagemaker:VolumeKmsKey` condition key.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerNoteBookEnforceEncryption",
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateNotebookInstance",
        "sagemaker:UpdateNotebookInstance"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "sagemaker:VolumeKmsKey": "true"
        }
      }
    }
  ]
}
```

- **Enforcing Studio Notebook EFS encryption** — The EFS storage encryption can be enforced using the `sagemaker:VolumeKmsKey` condition key.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerStudioEnforceEncryption",
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateDomain"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "sagemaker:VolumeKmsKey": "true"
        }
      }
    }
  ]
}
```

- **Enforcing job encryption** — Similarly, encryption for the SageMaker training job, processing job, transform job, and hyperparameter tuning job can be enforced using the `sagemaker:VolumeKmsKey` condition key.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerJobEnforceEncryption",
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateHyperParameterTuningJob",
        "sagemaker:CreateProcessingJob",
        "sagemaker:CreateTrainingJob",
        "sagemaker:CreateTransformJob"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "sagemaker:VolumeKmsKey": "true"
        }
      }
    }
  ]
}
```

- **Enforcing inter-container traffic encryption** — For extremely sensitive distributed model training job and tuning job, the `sagemaker:InterContainerTrafficEncryption` condition key can be used to encrypt inter-container traffic.

  **Note**
  The training speed will be negatively impacted when this is enabled.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerEnforceInterContainerTrafficEncryption",
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateHyperParameterTuningJob",
        "sagemaker:CreateTrainingJob"
      ],
      "Resource": "*",
      "Condition": {
        "Bool": {
          "sagemaker:InterContainerTrafficEncryption": "false"
        }
      }
    }
  ]
}
```

## Controlling data egress

- **Enforcing deployment in VPC** — To route traffic from SageMaker to access resources in a VPC, `sagemaker:VpcSubnets` and `sagemaker:VpcSecurityGroupIds` can be used to configure VPC and security group to manage the traffic.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerEnforceVPCDeployment",
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateHyperParameterTuningJob",
        "sagemaker:CreateModel",
        "sagemaker:CreateNotebookInstance",
        "sagemaker:CreateProcessingJob",
        "sagemaker:CreateTrainingJob"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "sagemaker:VpcSubnets": "true",
          "sagemaker:VpcSecurityGroupIds": "true"
        }
      }
    }
  ]
}
```

- **Enforcing Network Isolation** — Networking traffic can be blocked for the algorithm container by using the `sagemaker:NetworkIsolation` condition key.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "NetworkIsolation",
    "Effect": "Deny",
    "Action": [
      "sagemaker:CreateHyperParameterTuningJob",
      "sagemaker:CreateTrainingJob"
    ],
    "Resource": "*",
    "Condition": {
      "Bool": {
        "sagemaker:NetworkIsolation": "false"
        }
      }
    }
  ]
}
```

- Restricting access to SageMaker API and runtime by IP address — You can restrict the IP address ranges for invoking different SageMaker APIs by using the `aws:SourceIp` condition key.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerRestrictToIp",
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateTrainingJob",
        "sagemaker:CreateHyperParameterTuningJob",
        "sagemaker:CreateModel",
        "sagemaker:CreateProcessingJob"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:NotIpAddress": {
          "aws:SourceIp": ["<public IP address>"]
        }
      }
    }
  ]
}
```

- Restricting Studio and notebook pre-signed URLs to IPs — Launching SageMaker Studio or SageMaker Notebook instance can be restricted by the `aws:SourceIp`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerStudioRestrictToIp",
      "Effect": "Deny",
      "Action": [
         "sagemaker:CreatePresignedNotebookInstanceUrl",
         "sagemaker:CreatePresignedDomainUrl"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:NotIpAddress": {
          "aws:SourceIp": ["<public IP address>"]
        }
      }
    }
  ]
}
```

## Disabling internet access

- **Disabling SageMaker Notebook internet access** — If you want to disable internet access when the notebook is created, you can use `sagemaker:DirectInternetAccess` to achieve this.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerPreventDirectInternet",
      "Effect": "Deny",
      "Action": "sagemaker:CreateNotebookInstance",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "sagemaker:DirectInternetAccess": [
            "Enabled"
          ]
        }
      }
    }
  ]
}
```

- **Disabling Studio Domain internet access** — For SageMaker Studio, the following condition key may be used to disable internet access from the Studio domain:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerPreventDirectInternetforStudio",
      "Effect": "Deny",
      "Action": "sagemaker:CreateDomain",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "sagemaker:AppNetworkAccessType": [
            "PublicInternetOnly"
          ]
        }
      }
    }
  ]
}
```

## Preventing privilege escalation

- **Disabling SageMaker Notebook root access** — AWS recommends disabling the root access to SageMaker Notebooks for the data scientists and ML engineers. The following policy prevents a user from launching a SageMaker Notebook if `RootAccess` is not disabled.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerDenyRootAccess",
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateNotebookInstance",
        "sagemaker:UpdateNotebookInstance"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "sagemaker:RootAccess": [
            "Enabled"
          ]
        }
      }
    }
  ]
}
```

## Enforcing tags

- **Requiring tag for API call in dev environment** - the following policy requires a “dev” environment tag to be attached to the SageMaker endpoint.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerRequireEnvTag",
      "Effect": "Deny",
      "Action": "sagemaker:CreateEndpoint",
      "Resource": "arn:aws:sagemaker:*:*:endpoint/*",
      "Condition": {
        "StringNotEquals": {
          "aws:RequestTag/environment": "dev"
        }
      }
    }
  ]
}
```

- **Requiring tag for Studio domains in data science accounts** - To ensure that administrators appropriately tag Studio domains, kernels, and notebooks on creation, you can use the following policy. For example, for developers in data science accounts inside an OU, a Studio created in these accounts should be tagged as follows.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
                "Sid": "RequireAppTag",
                "Effect": "Deny",
                "Action": [
                    "sagemaker:CreateDomain",
                ],
                "Resource": "*",
                "Condition": {
                    "StringNotLike": {
                        "aws:RequestTag/Project": "data_science"
                    }
                }
            }
        ]
}
```

## Controlling cost

- **Enforcing instance type for a SageMaker Notebook instance** — The following policy ensures that only the listed instances types can be used to create a notebook instance.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerLimitInstanceTypes",
      "Effect": "Deny",
      "Action": "sagemaker:CreateNotebookInstance",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringNotLike": {
          "sagemaker:InstanceTypes": [
            "ml.c5.xlarge",
            "ml.m5.xlarge",
            "ml.t3.medium"
          ]
        }
      }
    }
  ]
}
```

- **Enforcing instance type for Studio Notebook instance** — The following policy helps enforce the type of instances used for SageMaker Studio notebook.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SageMakerAllowedInstanceTypes",
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateApp"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringNotLike": {
          "sagemaker:InstanceTypes": [
            "ml.c5.large",
            "ml.m5.large",
            "ml.t3.medium"
          ]
```

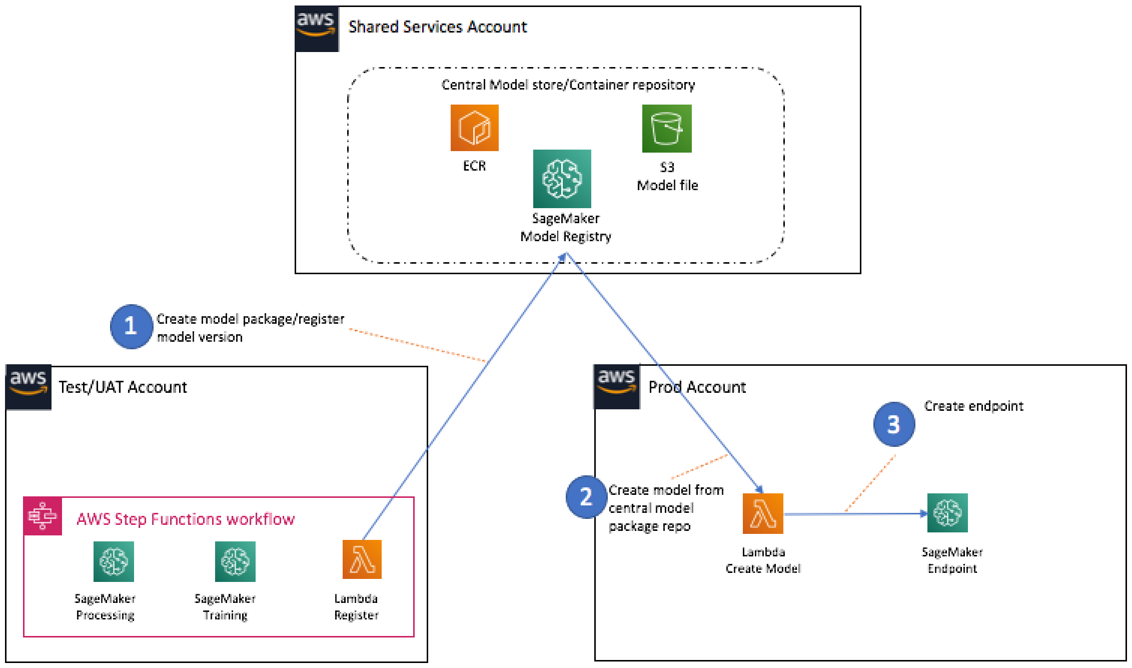
```
            }
          }
        }
      ]
    }
```

## Model inventory management

Model inventory management is an important component of model risk management (MGM). All models deployed in production need to be accurately registered and versioned to enable model lineage tracking and auditing. SageMaker provides a model registry feature for cataloging models for production and managing different model versions. With SageMaker model registry, you can also associate metadata with a model, such as training metrics, model owner name, and approval status.

There are several approaches for managing the model inventory across different accounts and for different environments. Following are two different approaches within the context of building a ML platform.

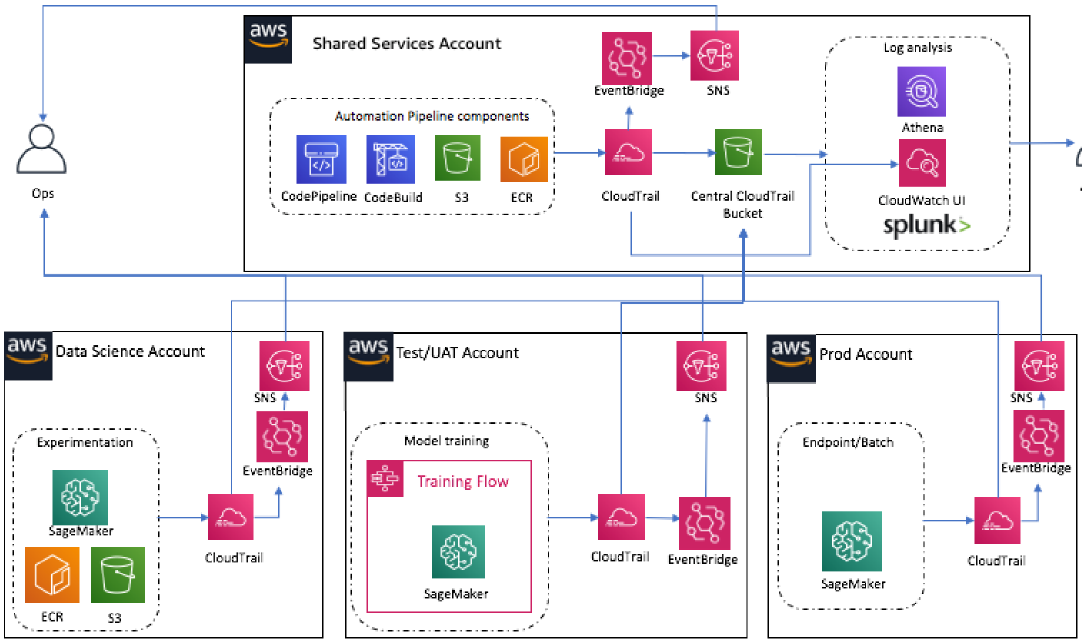
- **Distributed model management approach** — With this approach, the model files are stored in the account / environment in which it is generated, and the model is registered in the SageMaker model registry belonging to each account. For example, each business unit can have its own ML UAT / Test account, and the models generated by the automation pipelines are stored and registered in the business unit's own UAT / Test account.
- **Central model management approach** — With this approach, all models generated by the automated pipelines are stored in the Shared Services account along with the associated inference Docker container images, and a model package group is created to track different versions of a model. When model deployment is required in the production account, create a model in the production account using a versioned model Amazon Resource Name (ARN) from the central model package repository and then deploy the model in the production environment.



*Audit trail management*

## Audit trail management

Operations against AWS services are logged by AWS CloudTrail, and log files are stored in S3. Access details such as Event Name, User Identity, Event Time, Event Source, and Source IP are all captured in CloudTrail.



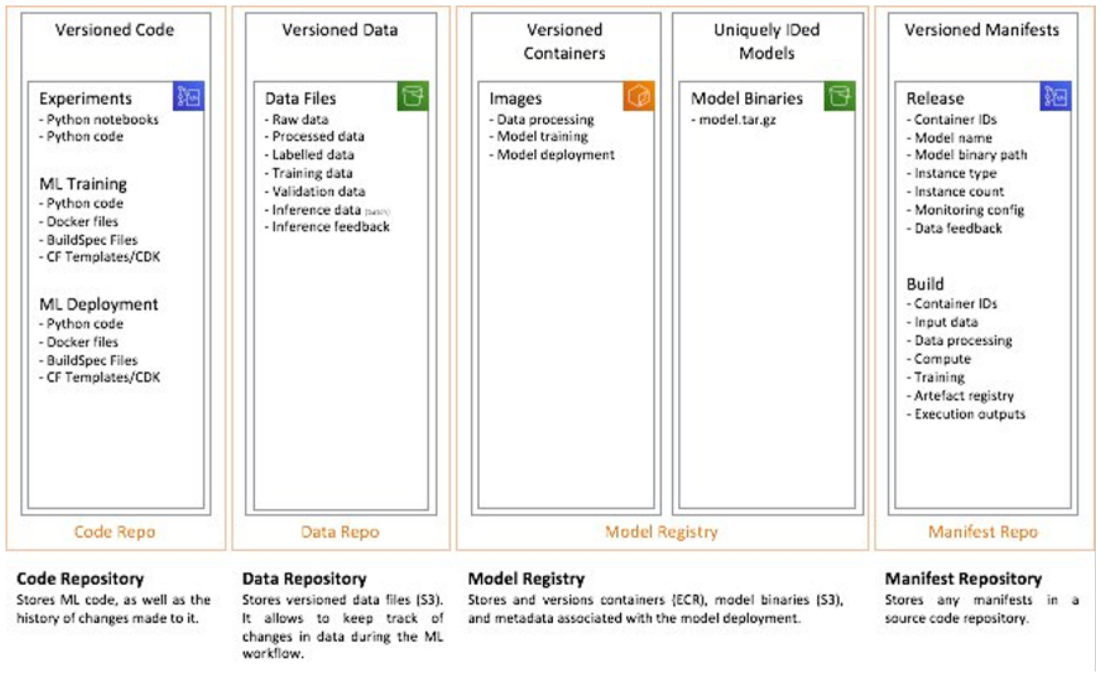
*Sample audit trail architecture*

CloudTrail provides features for accessing and viewing CloudTrail events directly in the console. CloudTrail can also integrate with log analysis tools such as Splunk to further processing and reporting.

SageMaker services such as notebook, processing job, or training job report the IAM roles assumed by these individual services against the different API events. To associate these activities with each individual user, consider creating a separate IAM role for each user for the different SageMaker services to assume.

## Data and artifacts lineage tracking

For regulated customers, tracking all the artifacts used for a production model is an essential requirement for reproducing the model to meet regulatory and control requirements. The following diagram shows the various artifacts that need to be tracked and versioned to recreate the data processing, model training, and model deployment process.



**Code Repository**
Stores ML code, as well as the history of changes made to it.

**Data Repository**
Stores versioned data files (S3). It allows to keep track of changes in data during the ML workflow.

**Model Registry**
Stores and versions containers (ECR), model binaries (S3), and metadata associated with the model deployment.

**Manifest Repository**
Stores any manifests in a source code repository.

*Artifacts needed for tracking*

- **Code versioning** — Code repositories such as GitLab, Bitbucket, and CodeCommit support versioning of the code artifacts. You can check-in / check-out code by commit ID. Commit ID uniquely identifies a version of source code in a repository.
- **Dataset versioning** — Training data can be version controlled by using a proper S3 data partition scheme. Every time there is a new training dataset, a unique S3 bucket / prefix can be created to uniquely identify the training dataset.

  DVC, an open-source data versioning system for machine learning, can track different versions of a dataset. The DVC repository can be created with a code repository such as GitHub and CodeCommit, and S3 can be used as the back-end store for data.

  Metadata associated with the datasets can also be tracked along with the dataset using services such as SageMaker Lineage Tracking.
- **Container versioning** — Amazon ECR uniquely identifies each image with an Image URI (repository URI + image digest ID). Images can also be pulled with repository URI + tag (if a unique tag is properly enforced for each image push). Additional tags can be added to track other metadata for source code repo URI and commit ID for traceability.
- **Training job versioning** — Each SageMaker training job is uniquely identified with an ARN, and other metadata such as hyperparameters, container URI, data set URI, and model output URI are automatically tracked with the training job.
- **Model versioning** — A model package can be registered in SageMaker using the URL of the model files in S3 and URI to the container image in ECR. An ARN is assigned to the model package to identify the model package uniquely. Additional tags can be added to the SageMaker model package with the model name and version ID to track different versions of the model.
- **Endpoint versioning** — Each SageMaker endpoint has a unique ARN, and other metadata, such as the model used, are tracked as part of the endpoint configuration.

# Infrastructure configuration change management

As part of the ML platform operation, Cloud Engineering / MLOps team needs to monitor and track resource configuration changes in core architecture components such as automation pipelines and model hosting environment.

AWS Config is a service that enables you to assess, audit, and evaluate the configuration of AWS resources. It tracks changes in CodePipeline pipeline definitions, CodeBuild projects, and CloudFormation stacks, and it can report configuration changes across timelines.

AWS CloudTrail can track and log SageMaker API events for the create, delete, and update operations for model registry, model hosting endpoint configuration, and model hosting endpoint to detect production environment changes.

# Container repository management

## Image Heirarchy Management

**Image hierarchy Management**

Most regulated customers have strict requirements around container patching, scanning, and hardening. Management of all container images in a consistent fashion may be challenging. If this is your situation, you will benefit from the following container image hierarchy:

- Public images are images for OS and/or a specific version of runtime environments such as Ubuntu or TensorFlow Docker images. These images are stored in public repositories such as ECR Public Gallery, Github, and Docker Hub.
- Base images are foundational images stored in the customer ECR instance. These images are typically built from public images with additional custom patches and OS-level hardening procedures. Base images are owned by the central team and used across an organization.
- Framework images are built on top of base images and should provide a stable version of the environment with a specific version of frameworks. For instance, platform container with the latest stable TensorFlow Serving framework configured. Framework images are typically also owned by the central team and available across the organization. Framework can also support dynamic loading of sources for flexibility.
- Application images are used by teams to create a specific environment such as copy sources or compiled binaries or model artifacts for distribution. These images are owned by application teams and should be based on vetted framework images where possible.

  **Organizing repositories** — Amazon ECR provides a number of features to help organize images in the repository.
- **Namespaces** — Amazon ECR supports namespaces, which allows the grouping of similar repositories into a common name. For example, each team or department can have its own namespace. This avoids potential conflicts with image naming, even if two departments use the exact same image names. In such a case, `department1/tensorflow-serving:latest` and `department2/tensorflow-serving:latest` are different images, as they are prefixed by a unique namespace.
- **Image tagging** — Container registries provide the functionality to mark any image with a particular tag. There are several tagging schemes applicable for different use cases:
  - **Stable tags** — These tags typically have a specific version of OS or environment. For example, a “`tf-serving:v2.3`” image where “`v2.3`” is a stable tag to designate a specific version of pre-installed TensorFlow framework. A stable version doesn’t mean that image will be frozen, as TensorFlow framework can have minor releases and patches going on. For that reason, it’s recommended to use stable tags for the base and/or platform images.
  - **Unique tags** — This type of tag is typically used when tagging application images, and it includes a reference to the GitHub repository branch and commit hash. This allows easily tracing images to the source code if needed.
- **Container security** — It is important to scan images for vulnerabilities and establish a continuous patching mechanism for all images. Consider incorporating these steps into the automated Docker image-build pipelines.

  Vulnerabilities in container software can be found on all levels: OS, packages, frameworks, or application code. It’s important to have an automatic way to consistently scan all container images on a recurrent basis.

Amazon ECR image scanning helps in identifying software vulnerabilities in your container images. Amazon ECR uses the Common Vulnerabilities and Exposures (CVEs) database from the open-source Clair project and provides you with a list of scan findings. You can review the scan findings for information about the security of the container images being deployed. For more information about Clair, see Clair on GitHub.

A customer may choose to use other 3rd party solutions, such as Aqua Container Security Platform, to scan for vulnerabilities before pushing containers to ECR.

With the proposed hierarchical approach for container images, it's easy to ensure that all images are patched regularly. It's recommended to rebuild base images with the latest OS-level patches at least every 30 days (or as soon as known vulnerabilities are fixed). All dependent platform images should also be rebuilt automatically after the patch is applied to base images. Application containers will be rebuilt on the next commit and will include patches from the upstream platform and base images.

- **Access Control** — Amazon ECR is fully integrated with the Identity and Access Management service to provide you with granular authorization controls. It supports two types of policies:
  - **Resource-based policies** grant usage permission to other accounts on a per-resource basis. You can also use a resource-based policy to allow an AWS service to access your Amazon ECR repositories.
  - **Identity-based policies** specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. These policies are applied to IAM roles.
- **Cross-account access** — The ECR repository, hosted in the Shared Services account, needs to provide access to users or services from other AWS accounts. Amazon ECR supports granting such permissions via IAM policies.

  For example, the following policy establishes access to ECR repository by a principal in another account.

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "SagemakerECRRepo",
      "Effect": "Allow",
      "Principal": {
        "AWS": "<AWS_SageMaker_Principal_ARN in other account>"
      },
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability",
        "ecr:PutImage",
        "ecr:InitiateLayerUpload",
        "ecr:UploadLayerPart",
        "ecr:CompleteLayerUpload"
      ]
    }
  ]
}
```

# Tag management for an ML platform

The tagging mechanism in AWS enables you to assign metadata information to virtually any AWS resource. Each tag is represented by a key-value pair. Tagging provides a flexible mechanism to categorize and manage cloud resources. Tags become especially crucial as AWS utilization grows.

Tags can serve many purposes. Common use cases of using tags include:

- **Business tags and cost allocation** — In this scenario, you use tags such as cost center, business unit, team, or project to track resource ownership and AWS spend, and create financial reports.
- **Tags for automation** — Tags are used to separate resources belonging to different projects and apply different automation policies (for example, shut down all project 1 resources on the weekend, but keep project 2 intact).
- **Security and risk management** — Apply different tags based on compliance or security policies. For example, use tags to separate sensitive data and apply different policies for it.

In an ML platform, tagging should be considered for the following resources:

- **Experimentation environment resources** — The SageMaker Notebook provision should be automated using CloudFormation script and tags such as owner name, owner id, dept, environment, and project should be automatically populated by the CloudFormation script. For other resources (for example, training job or end point) that are programmatically provisioned by data scientists through SDK / CLI, preventive guardrails should be implemented to ensure that the required tags are populated.
- **Pipeline automation resources** — Resources automated by the automation pipeline should be probably tagged. Pipeline should be automatically provisioned using CloudFormation scripts and tags such as owner name/id, environment, and project should be automatically populated by the CloudFormation script. These resources include CodePipeline pipeline run, CodeBuild job, Step Functions run, SageMaker processing job / training job, SageMaker models, SageMaker endpoint, Lambda function, and SageMaker.
- **Production resources** — Resources such as SageMaker endpoints, model artifacts, and inference images should be tagged.

**Tag example 1** — Enforce tags with a specific pattern when creating new AWS resources.

You may want to enforce specific tagging patterns. For example, the following policy defines that a new SageMaker endpoint can be created if tag "team" is present with one of 3 possible values: "`alpha`", "`bravo`", and "`charlie`".

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateSageMakerEndPoint",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateEndpointConfig",
                "sagemaker:CreateEndpoint"
            ],
            "Resource": [
                "arn:aws:sagemaker:*:*:endpoint-config/*",
                "arn:aws:sagemaker:*:*:endpoint/*"
            ],
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "aws:RequestTag/team": [
                        "alpha",
                        "bravo",
                        "charlie"
                    ]
                }
            }
        }
    ]
}
```

**Tagging example 2** — Automatically delete any untagged resources.

Users typically mandate tagging of all AWS resources and consider untagged resources as non-compliant. They may want to apply remediation actions, such as deletion of untagged resources or placing it under quarantine. The simplest way to implement this automatic remediation of untagged resources is via AWS Config, using the managed rule "`required-tags`". This rule automatically applies to customer-defined resource types and checks whether customer resources have the required tags. For example, you can check whether your Amazon EC2 instances have the `CostCenter` tag. For more information, see required-tags.

**Tagging example 3** — Controlling access to AWS resources.

In many cases, tags provide a powerful and yet flexible way to manage access to various resources. This is an example of identity policy, which blocks access to any S3 objects unless the object has a "`security`" tag with the value "`shared`".

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3ObjectAccess",
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/security": "shared"
                }
            }
        }
    ]
}
```

For more information, see the *Tagging Best Practices* whitepaper.

# Conclusion

Enterprises often struggle getting started with their first enterprise-grade ML platform, because of the many components of an ML platform architecture and the additional complexity around data science, data management, and model and operational governance.

This whitepaper provides a comprehensive view of the various architecture components in a secure ML platform. It also provides implementation guidance on building an enterprise ML platform, such as experimentation environment, automation pipeline, and production model serving, from scratch, using AWS services.

You can follow the architecture patterns, code samples, and best practices in this paper to help you plan the architecture and build a secure ML platform.

# Document history and contributors

To be notified about updates to this whitepaper, subscribe to the RSS feed.

| update-history-change | update-history-description | update-history-date |
|---|---|---|
| Initial publication (p. 50) | First published. | May 11, 2021 |

## Contributors

Contributors to this document include:

- David Ping, Principal ML Solutions Architect and Sr. Mgr., AI/ML Solutions Architecture
- Stefan Natu, Principal AI/ML Solutions Architect
- Qingwei Li, Sr. AI/ML Solutions Architect
- Saeed Aghabozori, Sr. AI/ML Solutions Architect
- Vadim Dabravolski, Sr. AI/ML Solutions Architect
- Simon Zamarin, AI/ML Solutions Architect
- Ibrahim Gabr, AI/ML Solutions Architect
- Amir Imani, AI/ML Solutions Architect

# Notices