Architecture Diagrams

DevOps and CI/CD Reference Architecture Diagrams



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

DevOps and CI/CD Reference Architecture Diagrams: Architecture Diagrams

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

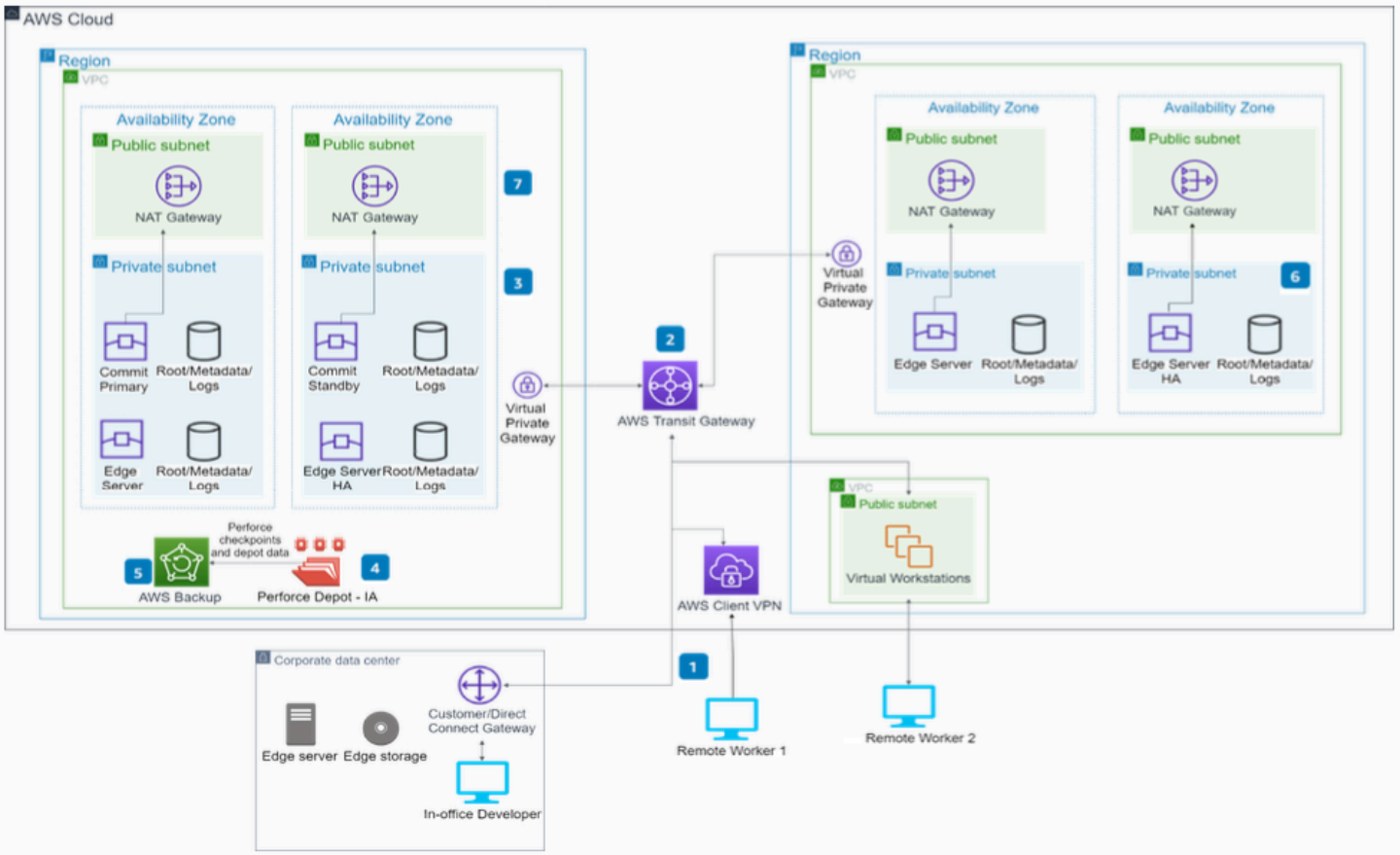
Distributed Perforce architecture on AWS 1
Distributed Perforce architecture on AWS diagram 1
Further reading 2
iOS build and test pipeline using Jenkins 3
iOS build and test pipeline using Jenkins diagram 3
Further reading 4
iOS CI/CD build and test pipeline 5
iOS CI/CD build and test pipeline diagram 5
Further reading 6
Pull request continuous integration 7
Pull request continuous integration diagram 7
Further reading 8
Unity build pipeline for iOS games 9
Unity build pipeline: iOS games on AWS Cloud diagram 9
Further reading 10
Diagram history 11

Distributed Perforce architecture on AWS

Publication date: **November 23, 2021 (Diagram history)**

With this architecture, you can deploy a hybrid multi-Region Perforce Helix Core architecture on AWS.

Distributed Perforce architecture on AWS diagram



The following steps describe the architecture:

1. Connect the corporate data center edge server to the AWS primary Region by using AWS Direct Connect or AWS Site-to-Site VPN. Choose based on bandwidth and connection stability needs. Connect remote users by using AWS Client VPN or virtual workstations on AWS.
2. AWS Transit Gateway connects VPCs and on-premises networks through a central hub-and-spoke model. It simplifies complex peering relationships and encrypts data in transit.

3. If your depot is less than 16 TB, run Perforce on Amazon EBS gp3 volumes. For depots larger than 16 TB, store the Perforce depot in Amazon Elastic File System. Use Amazon EFS Standard-Infrequent Access for cost optimization.
4. AWS Backup handles Amazon EFS backups. If you run Perforce on Amazon EBS only, Amazon EBS snapshots are the standard backup mechanism.
5. Edge Server high availability is not required, depending on recovery point objective and recovery time objective. Restoring from an Amazon EBS snapshot is a slower but more cost-effective solution.
6. Use a NAT gateway so that instances in a private subnet can connect to services outside your VPC. External services cannot initiate a connection with those instances.
7. Perforce commit-edge architecture offers the best overall performance with most commands running locally. The primary and replica/high availability servers run in separate Availability Zones for further high availability.

Further reading

For additional information, see the following resources:

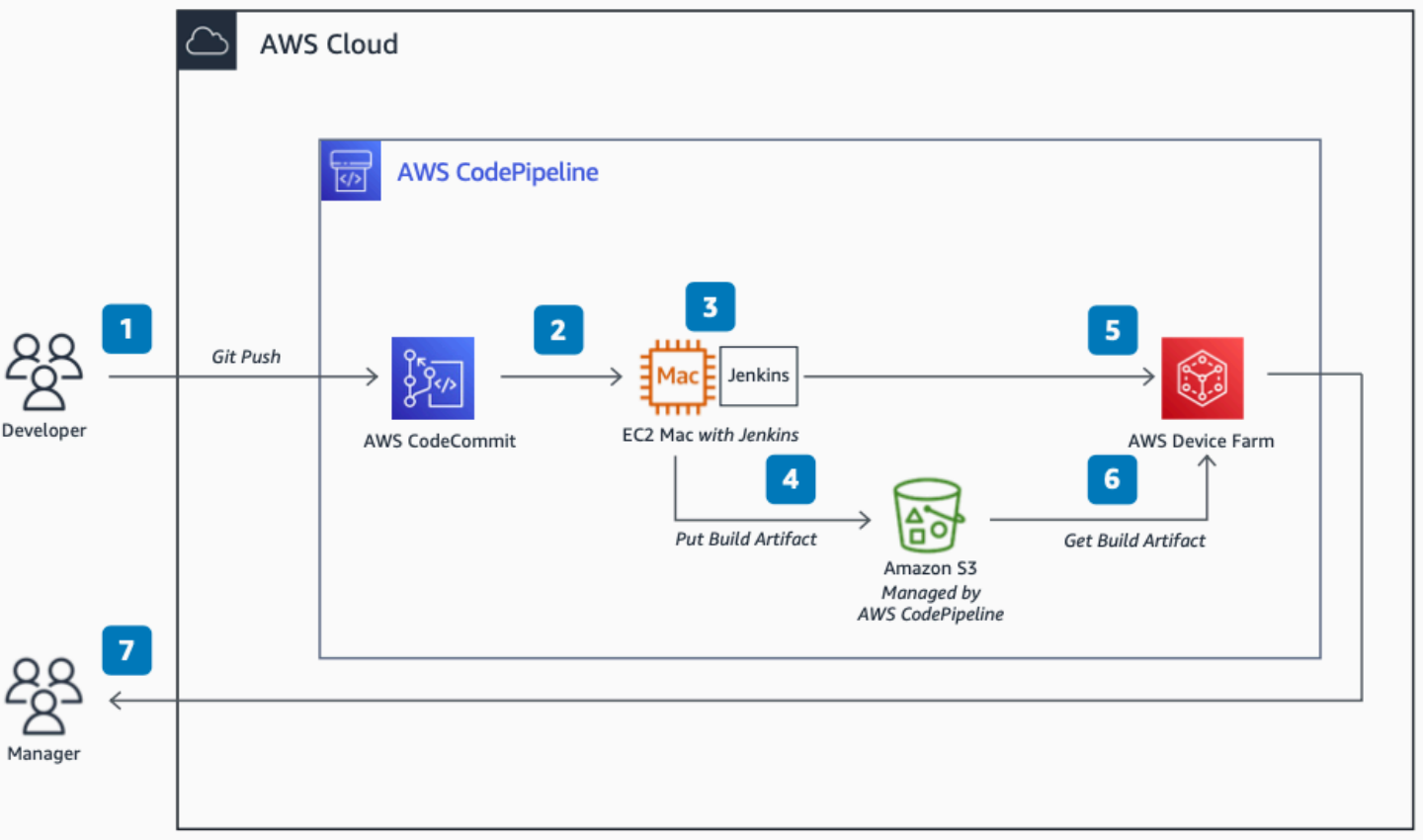
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected
- AWS Direct Connect product page
- AWS Game Tech product page

iOS build and test pipeline using Jenkins

Publication date: **November 23, 2021 (Diagram history)**

With this architecture, you can automate building and testing iOS apps by using Jenkins, Amazon Elastic Compute Cloud Mac instances, and AWS Device Farm.

iOS build and test pipeline using Jenkins diagram



The following steps describe the architecture:

1. A developer initiates a build and test activity in AWS CodePipeline by pushing a code change to AWS CodeCommit.
2. CodePipeline detects the change in CodeCommit and creates a job for Jenkins.
3. Jenkins on Amazon EC2 Mac uses the CodePipeline plugin to poll CodePipeline, downloads the source, and triggers the build project.

4. The Amazon EC2 Mac instance builds the artifacts (as .ipa files). The CodePipeline plugin for Jenkins compresses and uploads the build artifacts to CodePipeline through Amazon Simple Storage Service.
5. CodePipeline runs the test stage on Device Farm to test the build artifacts (.ipa files) on multiple real devices.
6. Device Farm automatically fetches the build artifacts from the Amazon S3 bucket, runs the test cases, and records the results.
7. Device Farm makes the test results, logs, and recordings available for review through the Device Farm console or through Amazon S3 pre-signed URLs.

Further reading

For additional information, see the following resources:

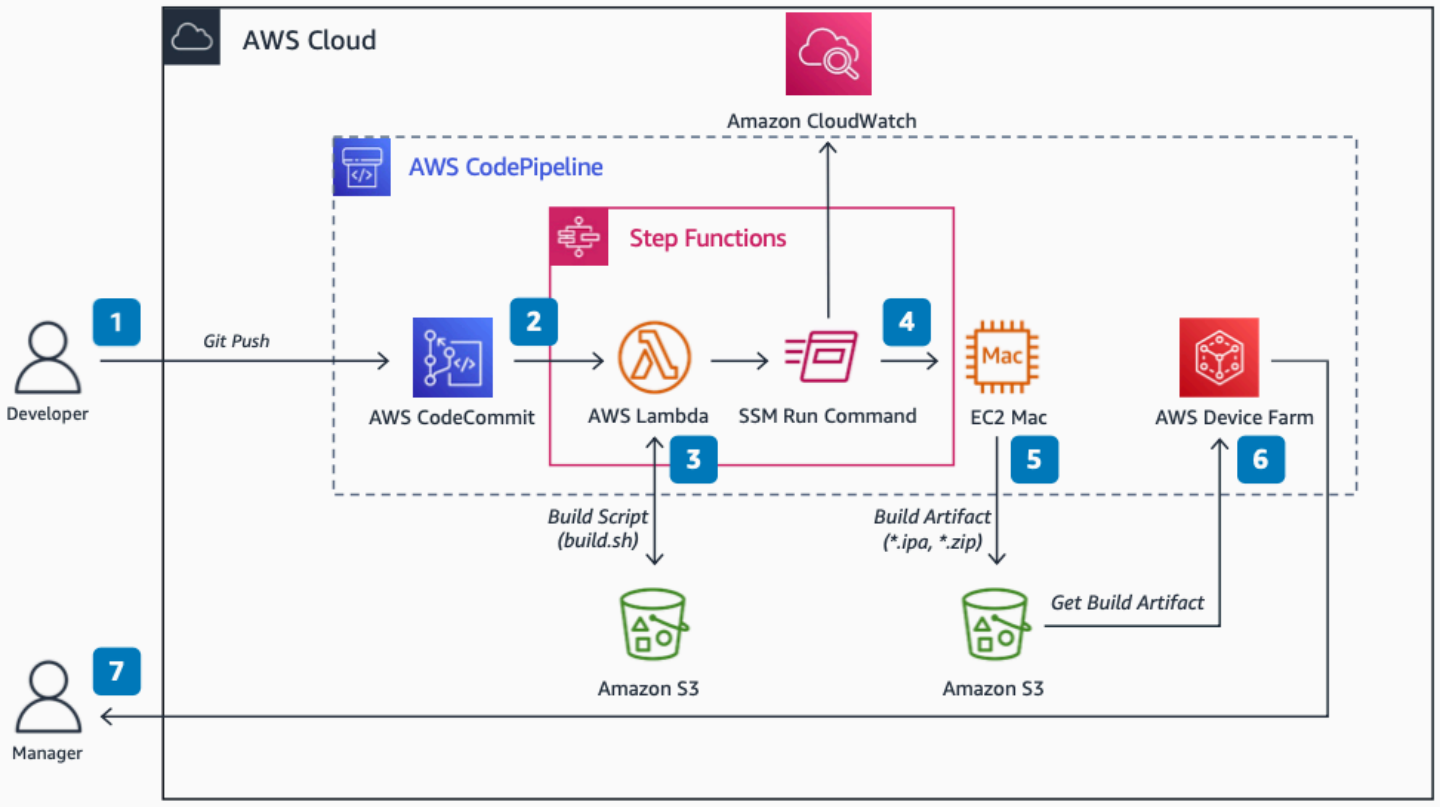
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected
- AWS CodePipeline product page

iOS CI/CD build and test pipeline

Publication date: **November 17, 2021 (Diagram history)**

With this architecture, you can automate building and testing iOS apps by using AWS CodePipeline, Amazon Elastic Compute Cloud Mac instances, and AWS Device Farm.

iOS CI/CD build and test pipeline diagram



The following steps describe the architecture:

1. A developer initiates a build or test activity in AWS CodePipeline by pushing a code change to AWS CodeCommit.
2. CodePipeline detects the change in CodeCommit and invokes AWS Step Functions to start the build process.
3. An AWS Lambda function loads required build scripts from an Amazon Simple Storage Service bucket and triggers an AWS Systems Manager Run Command.
4. Build scripts initiate an iOS build on the Amazon Elastic Compute Cloud Mac instance and put the build artifacts in an Amazon S3 bucket.

5. The Systems Manager Run Command runs the build scripts on an Amazon EC2 Mac instance and sends run logs to Amazon CloudWatch.
6. CodePipeline detects the built artifacts in Amazon S3 and starts Device Farm to test the app on multiple real devices.
7. Device Farm generates test results, logs, and recordings. You can review them through the Device Farm console or through Amazon S3 pre-signed URLs.

Further reading

For additional information, see the following resources:

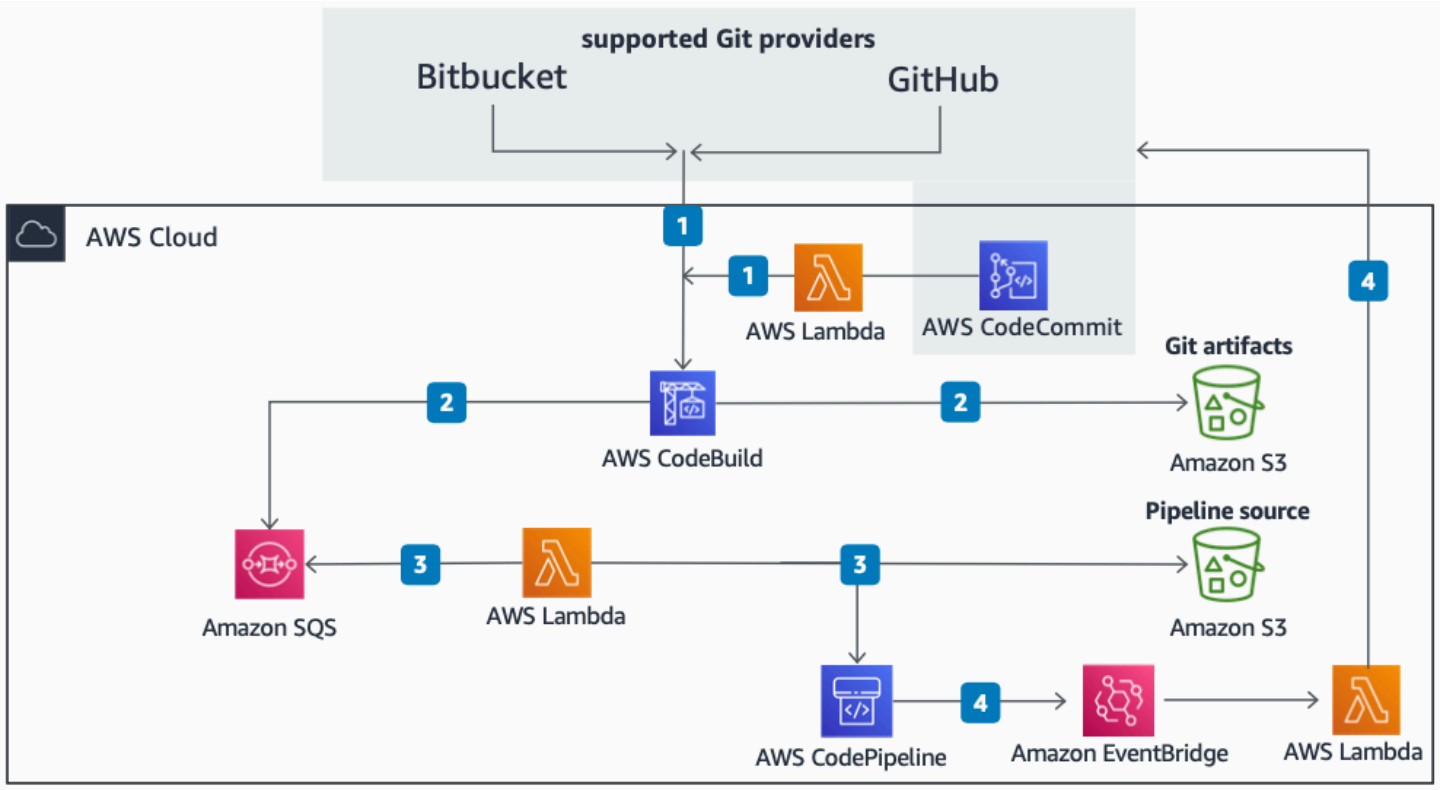
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected
- AWS CodePipeline product page

Pull request continuous integration

Publication date: **July 14, 2022 (Diagram history)**

With this architecture, you can build a continuous integration pipeline for Git pull requests by using AWS CodeBuild, AWS CodePipeline, and AWS Lambda.

Pull request continuous integration diagram



The following steps describe the architecture:

1. Webhooks connect supported Git providers (such as GitHub or Bitbucket) and AWS CodeBuild. The Git provider sends events when a PR is created or updated. For AWS CodeCommit, an AWS Lambda function triggers the build.
2. CodeBuild accepts Git PR events and compresses the PR codebase in an archive. It stores this archive in the Git Artifacts Amazon S3 bucket. CodeBuild then posts a message on an Amazon Simple Queue Service queue about a fresh codebase ready for processing.
3. An Lambda function polls the Amazon SQS queue and consumes the message. It starts the CI process and moves the new codebase to the Pipeline Source Amazon S3 bucket. The Lambda

function checks automation prerequisites. It also removes obsolete codebases from the backlog when updates target an existing PR.

4. AWS CodePipeline orchestrates the CI workflow. It validates the codebase, runs unit tests, builds artifacts, creates a preview environment, and runs integration and end-to-end (E2E) tests. Amazon EventBridge monitors pipeline stage changes. An Lambda function updates the PR based on status events (success, in progress, failed, completed).

Further reading

For additional information, see the following resources:

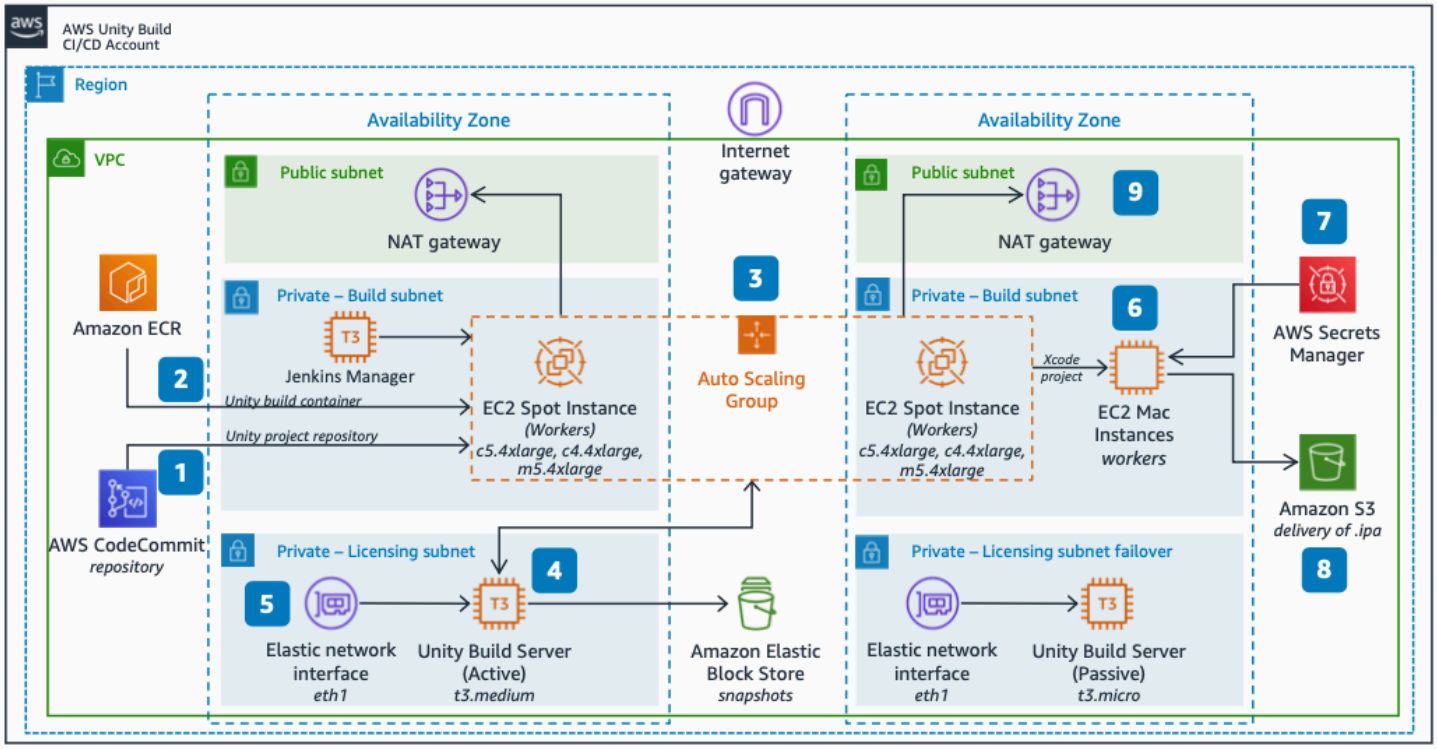
- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected
- AWS CodePipeline product page

Unity build pipeline: iOS games on AWS Cloud

Publication date: **November 24, 2021 (Diagram history)**

With this architecture, you can build Unity-based games for iOS in the AWS Cloud with Jenkins by using a two-stage pipeline on Amazon Elastic Compute Cloud Spot Instances and Amazon EC2 Mac instances.

Unity build pipeline: iOS games on AWS Cloud diagram



The following steps describe the architecture:

1. Source code is stored in an AWS CodeCommit repository. Jenkins pulls it on a build start.
2. A Unity container image is pulled from Amazon Elastic Container Registry and deployed by the Jenkins Docker agent on a worker node.
3. The first build stage runs on Amazon EC2 Spot Instances. It generates an Xcode project from Unity source code. These instances are placed into an Auto Scaling group for scalability and redundancy.
4. A Unity Build Server vends floating licenses to workers in the Auto Scaling group.

5. The Unity license binds to the Amazon EC2 instance's Ethernet interface MAC address. An elastic network interface preserves the MAC address if the instance is recreated.
6. The resulting Xcode project transfers to a Jenkins worker on an Amazon EC2 Mac Instance. This worker finalizes and signs the build and exports an .ipa file.
7. Certificates, private keys, and provisioning profiles are stored in AWS Secrets Manager. The Mac dynamically pulls them during a build.
8. An .ipa archive file is exported as a Jenkins artifact and stored in an Amazon S3 bucket.

Further reading

For additional information, see the following resources:

- AWS Architecture Icons
- AWS Architecture Center
- AWS Well-Architected
- AWS Game Tech product page

Diagram history

The following table describes updates to the reference architecture diagrams in this guide. To be notified about updates, subscribe to the RSS feed.

Change	Description	Date
Pull request continuous integration	Reference architecture diagram first published.	July 14, 2022
Unity build pipeline: iOS games on AWS Cloud	Reference architecture diagram first published.	November 24, 2021
iOS build and test pipeline using Jenkins	Reference architecture diagram first published.	November 23, 2021
Distributed Perforce architecture on AWS	Reference architecture diagram first published.	November 23, 2021
iOS CI/CD build and test pipeline	Reference architecture diagram first published.	November 17, 2021

Note

To subscribe to RSS updates, you must have an RSS plugin enabled for the browser you are using.