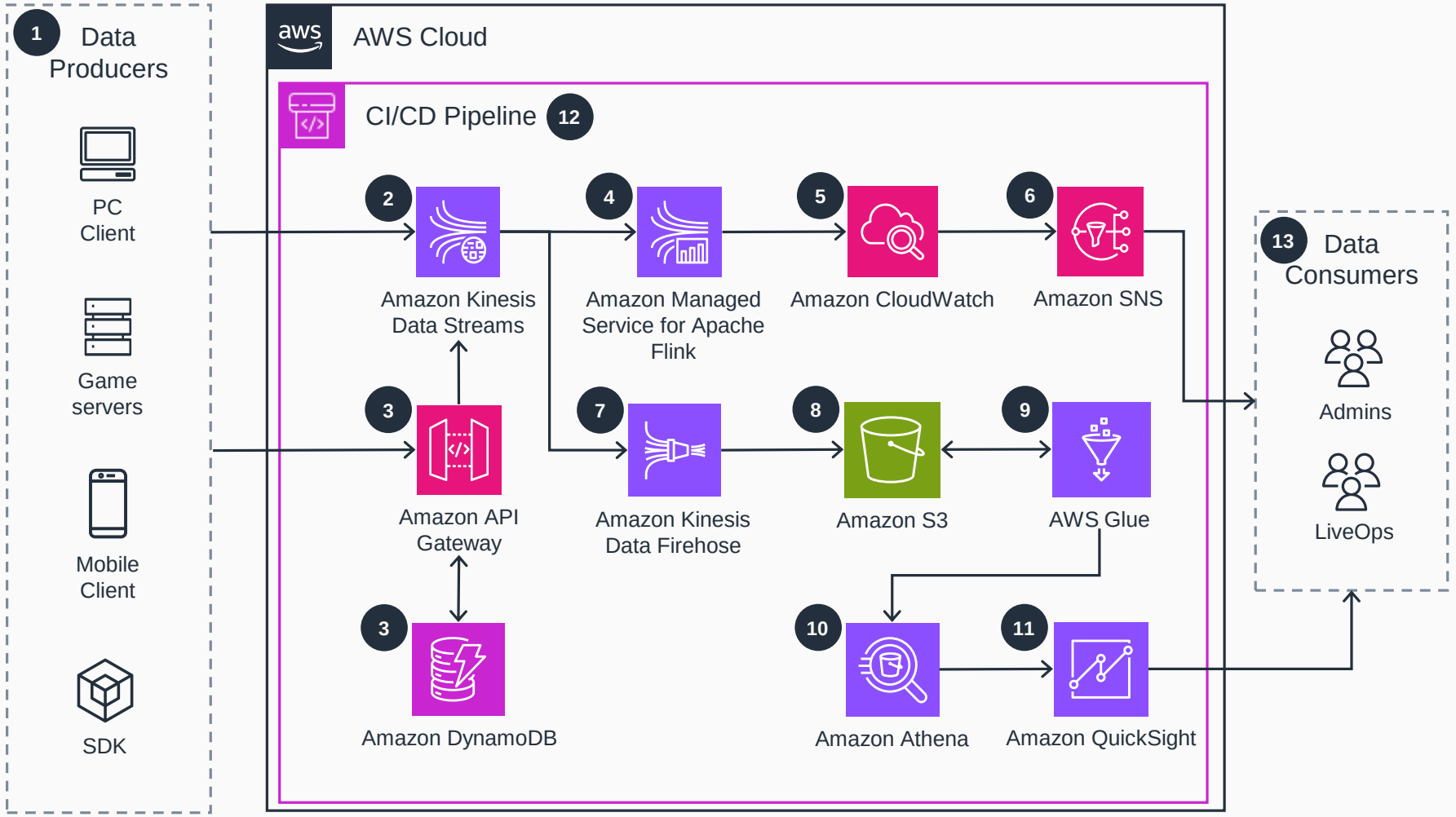
Guidance for Game Analytics Pipeline on AWS

Architecture

This architecture diagram shows a modernized DataOps pipeline for centralized game analytics on AWS.



1. Send game telemetry events to the AWS Cloud from data producers.
2. Capture streaming data from the game with **Amazon Kinesis Data Streams**, and process data in near real-time with **Amazon Kinesis Data Firehose** and **Amazon Managed Service for Apache Flink**.
3. Provide REST API endpoints with **Amazon API Gateway** to register data producers. Store game configurations and API access keys in **Amazon DynamoDB**.
4. Capture streaming event data in **Amazon Managed Service for Apache Flink**. Publish custom metrics in **Amazon CloudWatch**.
5. Create operational dashboards and alarms from custom metrics in **CloudWatch**.
6. Deliver critical alarm notifications to data consumers with **Amazon Simple Notification Service (Amazon SNS)**.
7. Process batched telemetry data with **Kinesis Data Firehose**.
8. Store both raw and processed telemetry data in **Amazon Simple Storage Service (Amazon S3)**.
9. Extract, transform, and load (ETL) stored telemetry data for analysis with **AWS Glue**.
10. Interactively query and analyze prepared data with **Amazon Athena**.
11. Visualize business intelligence (BI) data with **Amazon QuickSight**.
12. Deploy and operationalize the codified application using a continuous integration and continuous deployment (CI/CD) pipeline.
13. Provide LiveOps with BI, data visualizations, and machine learning (ML) capabilities from game telemetry data to generate key business insights..

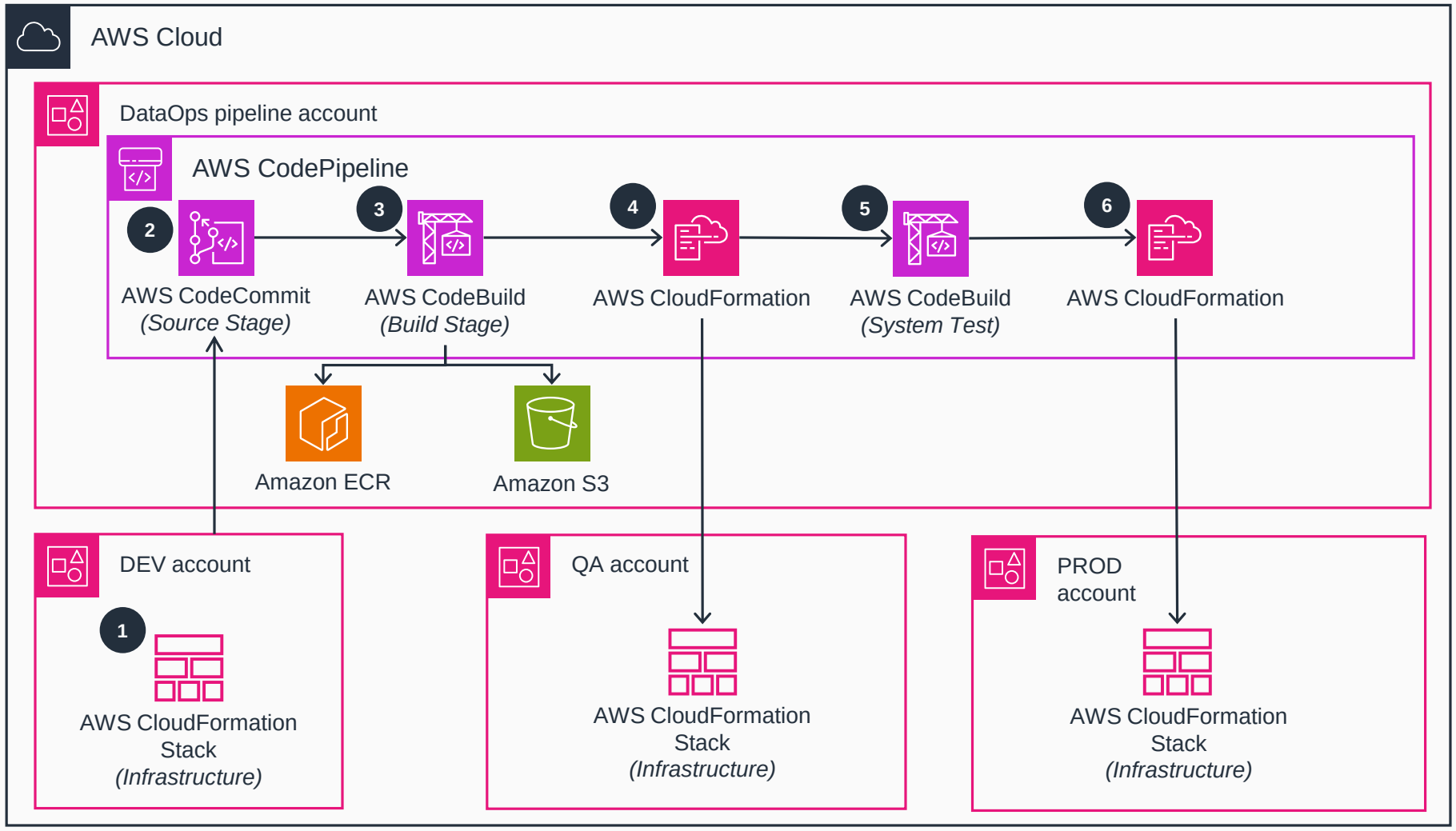
Reviewed for technical accuracy November 14, 2023

AWS Reference Architecture

Guidance for Game Analytics Pipeline on AWS

DataOps CI/CD Pipeline

This architecture diagram shows the DataOps CI/CD pipeline for centralized game analytics on AWS.



1. Build and test the codified infrastructure using the **AWS Cloud Development Kit (AWS CDK)** to synthesize an **AWS CloudFormation** template.
2. Initiate the CI/CD pipeline when infrastructure code changes are committed to the **AWS CodeCommit** repository.
3. Store compiled infrastructure assets, such as a Docker container and **CloudFormation** templates, in **Amazon Elastic Container Registry (Amazon ECR)** and **Amazon S3**.
4. Deploy the infrastructure for integration and system testing into the quality assurance (QA) AWS account using the **CloudFormation** Stack.
5. Run automated testing scripts to verify that the deployed infrastructure is functional inside an **AWS CodeBuild** project.
6. Deploy the tested infrastructure into the Production (PROD) AWS account using the **CloudFormation** Stack.

Reviewed for technical accuracy November 14, 2023

AWS Reference Architecture