Implementation Guide

Guidance for Deploying a Prebid Server on AWS

Guidance for Deploying a Prebid Server on AWS: Implementation Guide

Table of Contents

Deploy a Prebid Server to manage ad auction requests with AWS infrastructure

Guidance for Deploying a Prebid Server on AWS helps customers deploy and operate Prebid Server, an open source solution for real-time ad monetization, in their own Amazon Web Services (AWS) environment. The solution enables customers with ad-supported websites to achieve scaled access to advertising revenue through a community of more than 180+ advertising platforms. Customers achieve full control over decision logic and access to transaction data, and realize AWS benefits like global scalability and pay-as-you-go economics. This solution hosts the Prebid Server executable program in a cost-effective way for a variety of request loads—5,000-100,000 requests per second (RPS)--and captures operational and transaction logs from each Prebid Server container for analysis and reporting.

This implementation guide provides an overview of Guidance for Deploying a Prebid Server on AWS, its reference architecture and components, considerations for planning the deployment, configuration steps for deploying the solution to the AWS Cloud.

This guide is intended for solution architects, business decision makers, DevOps engineers, data scientists, and cloud professionals who want to implement Guidance for Deploying a Prebid Server on AWS in their environment.

Use this navigation table to quickly find answers to these questions:

If you want to . . .	Read . . .
Know the cost for running this solution. The baseline estimated cost for running this solution in the US East (N. Virginia) Region is approximately **$241.50 a month** when no load is running on the solution. See the Cost section for several impression-per-second and RPS scenarios.	Cost
Understand the security considerations for this solution.	Security
Know how to plan for quotas for this solution.	Quotas

If you want to . . .	Read . . .
Know which AWS Regions are supported for this solution.	Supported AWS Regions
Access the source code and use the AWS Cloud Development Kit (AWS CDK) to deploy the solution.	GitHub repository
Access the Prebid Server open source project.	GitHub repository

Features and benefits

Guidance for Deploying a Prebid Server on AWS provides the following features:

Prebid Server purpose built for AWS infrastructure

Deploy Prebid Server in a scalable and cost-efficient manner. It provides the end-to-end infrastructure to host a Prebid Server with production-grade availability, scalability, and low-latency for a variety of request loads (documented up to 100,000 RPS).

Built-in observability

Observability is available throughout the infrastructure. This includes operational resource metrics, alarms, runtime logs, and business metrics.

Decrease time to market

This solution uses a deployment template to establish the necessary infrastructure to get customers running within days instead of months or weeks.

Ownership of all operational and business data

All data from Prebid Server metrics extract, transform, and load (ETL) to AWS AWS Glue Data Catalog for seamless integration with various clients, such as Amazon Athena, Amazon Redshift, and Amazon SageMaker AI.

Use cases

Move ad-supported workflows to a dedicated, server-to-server (S2S) architecture

S2S header bidding moves auction processing and decisioning to dedicated servers, which improves page performance and protects advertising revenue.

Reduce operational complexity

Focus more on business insights and less on operating geographically distributed, high-volume, low-latency architectures — a challenging task even for well-resourced organizations.

Create complete infrastructure

This solution uses a deployment template to establish the necessary infrastructure, including load balancing, distribution, VPC resources, firewalls, data pipelines, and a container with the latest version of Prebid Server to get customers running within days instead of months or weeks.

Concepts and definitions

This section describes key concepts and defines terminology specific to this solution:

custom headers

Amazon CloudFront can be configured to add custom headers, which enable you to send and gather information from your origin that you don’t get with typical viewer requests.

header bidding

Header bidding is a process that enables publishers to capture bids for ad units from demand sources that might otherwise have been missed. By implementing header bidding, a publisher can gather bids from multiple sources that will then compete directly with bids from the ad server

Prebid Server

Prebid Server is an open source solution for server-to-server header bidding.

wrapper

A wrapper is a type of Software Development Kit (SDK) for advertising. A wrapper provides browser code to consolidate multiple header bidding partners into a unified framework. It simplifies the management of ad placements and real-time bidding across different ad networks, optimizing ad revenues by allowing simultaneous bids on inventory.

Note

For a general reference of AWS terms, see the AWS Glossary.

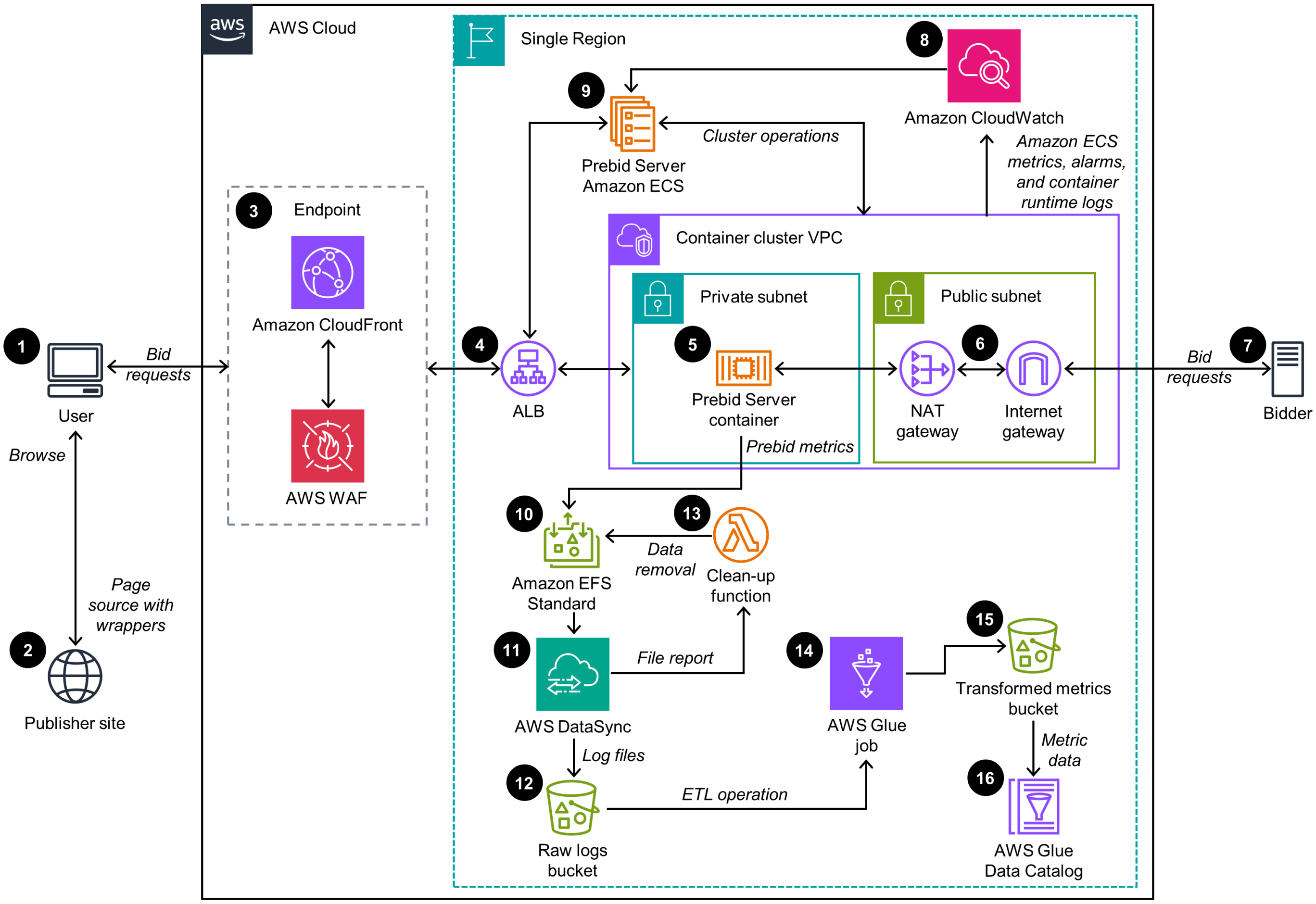
Architecture overview

This section provides a reference implementation architecture diagram for the components deployed with this solution and a summary of AWS Well-Architected design considerations.

Architecture diagram

Deploying this solution with the default parameters deploys the following components in your AWS account.

Guidance for Deploying a Prebid Server on AWS architectural overview



Note

AWS CloudFormation resources are created from AWS Cloud Development Kit (AWS CDK) constructs.

The high-level process flow for the solution, including the components deployed with the AWS CDK template, is as follows:

1. A user browses to a page on a website that hosts ads.
2. The publisher site returns the page source to the browser with resources, and one or more script modules (also called wrappers) that enable real-time bid requests and responses for ads of given dimensions, types, topics, and other criteria.
3. The bid requests are received from the browser at the Amazon CloudFront endpoint integrated with AWS Web Application Firewall (AWS WAF) for entry into the solution. This step helps validate legitimate traffic from malicious requests, such as penetration or denial-of-service attempts. Traffic can be received here as HTTP or HTTPS.
4. The request is forwarded to Application Load Balancer (ALB). ALB determines which container running Prebid Server in the cluster is at a utilization level that can accept more requests. ALB has a network interface on the public internet and one in each private subnet where containers are hosted within Amazon Virtual Private Cloud (Amazon VPC).
5. The request arrives at an Amazon Elastic Container Service (Amazon ECS) container, is parsed and validated, and requests to different bidding services are sent concurrently to the internet through the default internet gateway.
6. The NAT gateway and internet gateway allow containers to initiate outbound requests to the internet and receive responses. These resources are primarily used for Prebid Server containers to request and gather bids for ad auctions.
7. Bidders receive one or more bid requests over the internet from a Prebid Server container. Bidders respond with zero or more bids for the various requests. The response, including the body of the winning creative(s), is sent back to the browser.
8. During normal operation, Amazon CloudWatch metrics are collected from various resources involved in handling requests and responses through the solution. As the load changes throughout the cluster, CloudWatch alarms are used to determine when to scale-out or scale-in the container cluster.
9. An ECS service definition (Prebid ECS service) is responsible for tracking the health of the cluster, performing scale-out and scale-in operations, and managing the collection of containers available for ALB. The Prebid ECS service uses AWS Fargate instances.

10Metrics log files for each container are stored to a shared Amazon Elastic File System (Amazon EFS) using NFS protocol. This file system is mounted to each Prebid Server container during start-up. A single metrics log file is written for a limited time and then closed and rotated, so that it can be included in the next stage of processing. EFS is treated as a temporary location as log data is generated and moved to longer-term storage on Amazon Simple Storage Service (Amazon S3) and into AWS Glue.

11. AWS DataSync replicates rotated log files from EFS to S3 on a recurring schedule. DataSync verifies each transferred file and provides a report of the completed work to an AWS Lambda function.
12. The `DataSyncLogsBucket` S3 bucket receives the replicated log files from EFS using the same folder structure. Log files arrive in this bucket as a result of the DataSync process.
13. The `delete_efs_files` Lambda function runs after the DataSync process completes in step 12 and removes transferred and verified log file data from EFS.
14. An AWS Glue job performs an ETL operation on the metrics data in the `DataSyncLogsBucket` S3 bucket. The ETL operation structures the metric data into a single database with several tables, partitions the physical data, and writes it to an S3 bucket.
15. The `MetricsEtlBucket` S3 bucket contains the metric log data transformed and partitioned through ETL. The data in this bucket is made available to AWS Glue clients for queries.
16. Many different types of clients use AWS Glue Data Catalog to access the Prebid Server metric data.

AWS Well-Architected design considerations

This solution uses the best practices from the AWS Well-Architected Framework, which helps customers design and operate reliable, secure, efficient, and cost-effective workloads in the cloud.

This section describes how the design principles and best practices of the Well-Architected Framework benefit this solution.

Operational excellence

This section describes how we architected this solution using the principles and best practices of the operational excellence pillar.

Perform operations as code - This solution's infrastructure is entirely specified using AWS Cloud Development Kit(CDK) v2.0 in Python 3.x and deployed as a CloudFormation template. Application logging and metric workflows are automated with Amazon EventBridge and Lambda.

Make frequent, small, reversible changes - This solution is designed to be customized by the end user, if desired. The solution can be forked from the GitHub repository into a customer's account, customized, rebuilt, hosted in a customer’s Amazon S3 buckets, and deployed via CDK. This process can be repeated iteratively to test changes to the default solution.

Use managed services - Operational burden is reduced through the use of Amazon ECS to automatically manage and scale application containers in response to client request traffic.

Security

This section describes how we architected this solution using the principles and best practices of the security pillar.

Implement a strong identity foundation - All interactions among resources created by the solution are secured using AWS Identity and Access Management (IAM) roles, policies, and signature V4 request signing. All credentials used to interact among resources are temporary, and typically have a lifetime of less than one hour.

Maintain traceability - Runtime logging by Lambda functions installed by the solution is sent to Amazon CloudWatch Logs and preserved with the default retention settings.

Apply security at all layers - Interactions among resources require permissions defined in the related resource’s IAM role. AWS WAF protects public application endpoints from common web exploits. Security groups restrict inbound and outbound traffic at the resource level within the customers Amazon VPC.

Protect data in transit and at rest - All data is encrypted in transit via TLS-protected API requests. All persistent resources are configured for encryption at rest. Application-level data is encrypted with AWS Key Management Service using customer managed keys.

Reliability

This section describes how we architected this solution using the principles and best practices of the reliability pillar.

Automatically recover from failure - The solution uses Amazon CloudWatch metrics and alarms are used to monitor the operation of the solution with the ability to notify users or other systems when thresholds are breached.

Scale horizontally to increase aggregate workload availability - Client traffic is horizontally scaled with Amazon Elastic Container Service, distributed across containers using Elastic Load Balancing.

Stop guessing capacity - Resource demand is automatically monitored with Amazon ECS, maintaining optimal resource levels to satisfy demand without over- or under-provisioning.

Performance efficiency

This section describes how we architected this solution using the principles and best practices of the performance efficiency pillar.

Go global in minutes - The AWS CDK stack can be used to create a stack in any compatible Region, with the ability to deploy multiple stacks in the same Region for testing and production.

Use serverless architectures - Amazon ECS uses AWS Fargate serverless deployment to manage container resources at cloud scale without the operational burden of managing physical servers.

Consider mechanical sympathy - Application metrics data is transformed, partitioned, and stored in Amazon S3 and AWS Glue in accordance with common data access patterns to improve query performance.

Cost optimization

This section describes how we architected this solution using the principles and best practices of the cost optimization pillar.

Adopt a consumption model - Serverless computing is used to only pay for consumed compute resources on Amazon ECS.

Sustainability

This section describes how we architected this solution using the principles and best practices of the sustainability pillar.

Maximize utilization - Managed services allow for optimal resource provisioning to ensure high utilization while minimizing idle resources to maximize the energy efficiency of the underlying hardware.

Use managed services - This solution uses managed services such as Fargate and Lambda, which share resources across a broad customer base and reduces the amount of infrastructure needed to support cloud workloads.

Regulatory compliance

Prebid Server (Java) is packaged in this solution without any pre-configured settings that affect the treatment of consumer preferences for data privacy. When you run Prebid Server with this solution you must configure it to work how you want it to. Settings, such as privacy and consent controls, jurisdiction of operation, and data enrichment functions are all determined by configurations that you set. We advise you to follow the guidance provided on docs.prebid.org.

Architecture details

This section describes the components and AWS services that make up this solution and the architecture details on how these components work together.

AWS services in this solution

AWS service	Description
Amazon CloudFront	**Core**. Serve client requests to Prebid Server application.
AWS DataSync	**Core**. Automate transfer of Prebid Server application logs and metrics from Amazon EFS to Amazon S3.
Amazon ECS	**Core**. Host and manage containerized Prebid Server application.
Amazon EFS	**Core**. Centralize storage of Prebid Server application logs and metrics across containers.
Elastic Load Balancing	**Core**. Provide high availability and automate scaling of Prebid Server application containers hosted on Amazon ECS.
Amazon EventBridge	**Core**. Send and receive messages between solution resources handling Prebid Server application metrics and logs.
AWS Glue	**Core.** Transform, catalog, and partition metrics data into Amazon S3 and AWS Glue Data Catalog.
AWS Identity and Access Management (IAM)	**Core**. Restricts solution resource permissions to least privilege access for security.

AWS service	Description
AWS KMS	**Core**. Encrypt and decrypt the data in Amazon S3.
AWS Lambda	**Core**. Facilitate deployment and deletion of the solution through Lambda-backed custom resources, cleaning archived log and metrics files from Amazon EFS after being moved to Amazon S3 for long term storage, and triggering AWS Glue.
Amazon S3	**Core**. Provide long term storage of Prebid Server application logs and metrics from Amazon EFS.
AWS Systems Manager	**Core**. Provide application-level resource monitoring and visualization of resource operations and cost data.
Amazon VPC	**Core**. Control network permissions between solution resources.
AWS WAF	**Core**. Provide layer of security around Amazon CloudFront.
AWS CloudTrail	**Supporting**. Track activity across solution S3 buckets and Lambda functions.
Amazon CloudWatch	**Supporting**. View logs and subscribe to alarms for AWS Lambda and AWS Glue.
Amazon Athena	**Optional**. Access AWS Glue Data Catalog and query the Prebid Server application metrics in Amazon S3.

CloudFront distribution

The solution uses Amazon CloudFront as the unified network entry point. It receives the incoming auction requests and handles outgoing responses. CloudFront speeds up the distribution of your content by routing each user request through the AWS backbone network to the edge location that can best serve your content. CloudFront provides a TLS endpoint for privacy of requests and responses in transit with the pubic internet. ALB is the configured origin for CloudFront. Direct access to ALB is restricted by using a custom header, enhancing security.

AWS WAF

AWS Web Application Firewall (AWS WAF) and AWS Shield Standard are used as a protection mechanism from Distributed Denial of Service (DDoS) attacks against the Prebid Server cluster. AWS WAF can activate one or more managed rule groups by default after extended testing including rules in the Baseline Rule Group and the IP Reputation Rule Group. You have the option to activate, purchase, or use existing rule subscriptions, or add regular expression or CIDR matching rules as needed.

> **Note**
>
> If you want to opt out of using CloudFront and AWS WAF and directly send requests to the ALB, see How to opt out.

Application Load Balancer (ALB)

ALB distributes incoming request traffic for Prebid Server through the cluster of containers. It provides a single entry point into the cluster and is the primary origin for the CloudFront distribution.

Amazon VPC

The Amazon Virtual Private Cloud (Amazon VPC) is configured with redundant subnets, routes, and NAT gateways. Security groups permit traffic to and from the subnets. The Amazon VPC contains the network interfaces for the Prebid Server container cluster nodes. It is configured for private IP addresses only and container networks configured within the Amazon VPC use the NAT gateway as a default route to the internet for communication.

Amazon ECS

Amazon Elastic Container Service (Amazon ECS) is a fully managed container orchestration service that helps you easily deploy, manage, and scale containerized Prebid Server application. These resources define the configuration, count, and thresholds to scale-out and scale-in the total container count in the ECS cluster. The ECS task and service resource define the operating environment for the cluster and thresholds for scaling and health. Scaling changes are based on CPU, process load, and network traffic (requests per target). For cost optimization, ECS uses a weighted combination of Fargate and Fargate Spot instances. There's a cost benefit to using more Fargate Spot instances, but the risk of unavailability goes up. You might find that after running the solution for a while that a different ratio is better for you.

Prebid Server container

This is a docker container that runs the open source Prebid Server and is hosted in Amazon Elastic Container Registry (Amazon ECR). The container differs from the open source project's default container in configuration settings for areas including log output to the Console and bidding adapter configuration settings.

Amazon EFS

The EFS file system is mounted and shared among all container instances in the ECS cluster. This file system is used for log capture (operational and metrics), and has the potential to be expanded to include shared configuration and storage related to more advertisement types (for example, video and mobile).

DataSync (EFS to S3)

DataSync is configurated to periodically move rotated log files from each Prebid Server container's EFS location to an equivalent location in the `DataSyncLogsBucket` S3 bucket. After each file is copied to S3 and verified, it is removed from the EFS file system through a clean-up Lambda function. Essentially, only actively written log files are retained on the EFS file system until the Prebid Server process closes it, rotates it, and starts a new file. Rotated log files are migrated with DataSync. Runtime logs are rotated every 24 hours or when reaching 100 MB. Metrics logs are rotated every one hour or when reaching 100 MB.

Glue ETL (Metrics processing)

AWS Glue is a serverless data integration service that makes it easy for analytics users to discover, prepare, move, and integrate data from multiple sources. You can use it for analytics, machine learning, and application development. It also includes additional productivity and data ops tooling for authoring, running jobs, and implementing business workflows. This resource is responsible for periodically processing new metrics log files in the `DataSyncLogsBucket` S3 bucket. The CSV-formatted metrics are transformed into several tables and partitioned. After ETL processing completes, the new data is available to clients through AWS Glue Data Catalog.

AWS Glue Data Catalog

AWS AWS Glue Data Catalog provides access for clients to the Prebid Server metric data through Athena or other compatible clients, such as Amazon SageMaker AI, Amazon QuickSight, and JDBC clients. Clients can query and view the Prebid Server metrics data, generate graphs, summaries or inferences using AI/ML.

Amazon CloudWatch

CloudWatch alarms monitor specific metrics in real-time and proactively notify AWS Management Console users when predefined conditions are met. This solution has several CloudWatch alarms to help monitor its health and performance. These alarms are enabled automatically when the AWS CDK stack is deployed. For details, see the CloudWatch Alarms section.

Note

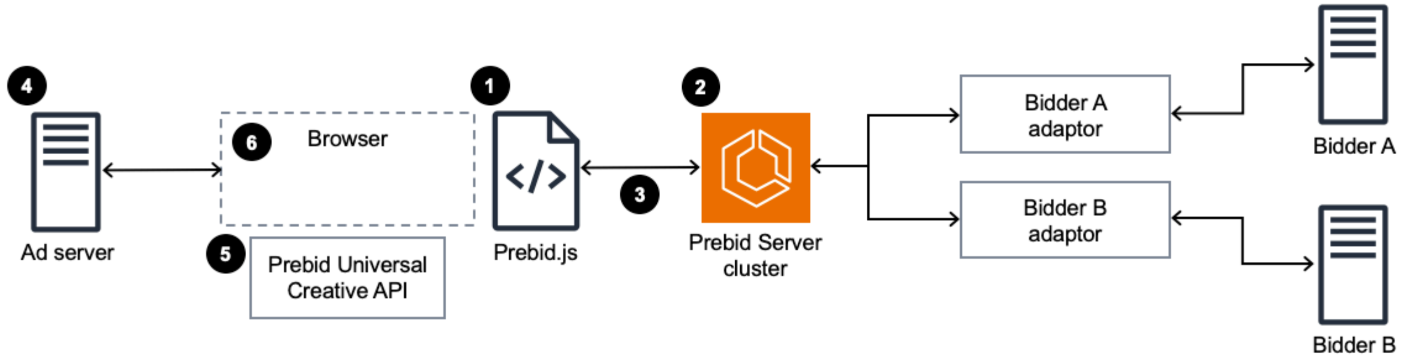
All resources are created in a single Region specified by the user except for CloudFront and AWS WAF. CloudFront is considered a global resource, and AWS WAF is always created in the `us-east-1` (N.Virginia) Region for configuration with CloudFront.

How Guidance for Deploying a Prebid Server on AWS works

Workflow of banner ads

From docs.prebid.org.

Workflow diagram of banner ads



1. `Prebid.js` is set up to run auctions for one or more bidders through `s2sConfig`.
2. Prebid Server parses the request and holds the auction.
3. The response, including the body of the winning creative(s), is sent back to the browser.
4. `Prebid.js` passes ad server targeting variables to the page, which forwards it to the ad server.
5. When a header bidding ad wins, the ad server responds to the page with the Prebid Universal Creative, which calls the render function in Prebid.js.
6. The render function displays the creative.

Amazon CloudFront, AWS WAF, and ALB

CloudFront is used as the entry point into the solution where the bid requests are received. CloudFront speeds up distribution of content through a worldwide network of data centers called edge locations. CloudFront ensures that end-user requests are served by the closest edge location. To prevent requests from bypassing the CDN and accessing the origin (ALB) directly, the solution uses CloudFront custom headers.

AWS WAF is a web application firewall that helps protect the application from common web exploits that could affect application availability, compromise security, or consume excessive resources. It is preconfigured with a set of standard AWS managed rules. AWS WAF is tightly integrated with CloudFront at the edge. Traffic can be received here as HTTP or HTTPS.

> **Note**
>
> If you want to opt out of using CloudFront and AWS WAF and directly send requests to the ALB, see How to opt out.

Prebid Server container

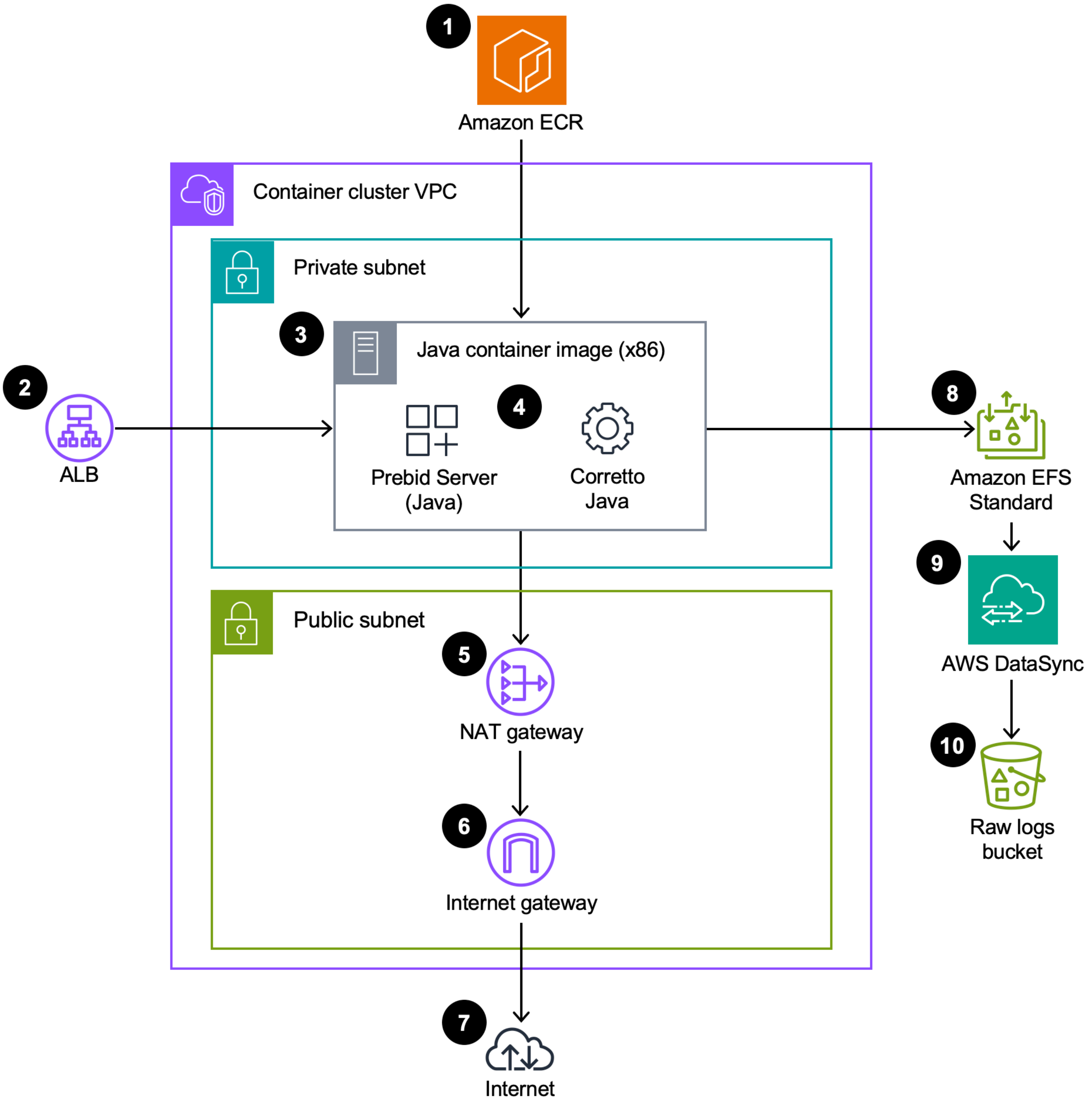
This container hosted in Amazon ECR runs the open source Prebid Server. The solution's default container image is a customized build of the Prebid Server implementation in Java. The Dockerfile for building this container image is located at `deployment/ecr/prebid-server/Dockerfile` and an accompanying configuration file is at `deployment/ecr/prebid-server/config.json`. This configuration file provides the version of the prebid server through the `GIT_TAG_VERSION` parameter.

The container differs from the open source project's default container in configuration settings for areas including file output and bidding adapter configuration settings. Alternatively, you can use the Go implementation of Prebid Server, or any build hosted in a container repository, such as Docker Hub or ECR Public Gallery. You will have the option of overriding the solution's default container.

> **Note**
>
> - If using a custom image, ensure it is compatible with the logging configuration of the container supplied with the solution to make use of the Prebid Metrics ETL.
> - Make changes to the Dockerfile as needed.

The following diagram shows a detailed view of the container contents, and the resources the container interacts with directly. Additionally, the container image supplied with the solution is stored in the ECR Public Gallery for AWS Solutions.

Prebid Server container diagram



1. Amazon ECR is a fully managed container registry for developers to store, manage, and deploy container images for public or private access.
2. Inbound auction requests come through Application Load Balancer. ALB distributes incoming application traffic across multiple ECS containers.
3. The Guidance for Deploying a Prebid Server on AWS container instances run in a defined environment.

4. The Java-based version of Prebid Server is configured to run with Amazon Corretto.
5. NAT gateway enables instances in a private subnet to connect to the internet but prevents the internet from initiating a connection with those instances. It is the default gateway for outgoing network packets from the private subnet within Amazon VPC.
6. The internet gateway connects a VPC with the public internet, allowing instances such as NAT gateways to send and receive traffic.
7. Auction requests are sent to bidders over the internet.
8. EFS Standard is a managed file storage service that provides a scalable file system for use with AWS Cloud services and on-premises resources. The EFS filesystem is used for capturing transaction and operational log data from each container via an NFS network connection.
9. DataSync is used as the ETL mechanism for moving data captured in raw log files on EFS to a structured and normalized layout in AWS Glue Data Catalog.
10. The `DataSyncLogsBucket` S3 bucket receives the migrated log data from the EFS filesystem. The Glue ETL process uses the contents of this bucket as source data.

ECS Fargate containers automatic scaling

Automatic scaling is the ability to increase or decrease the desired count of tasks in your Amazon ECS service automatically. The service is configured to use target tracking scaling policies, which increase or decrease the number of tasks that your service runs. These are based on a target value for a specific metric provided by Application Auto Scaling, such as average CPU utilization, memory utilization, and average request count per target. These values are set in the `source/infrastructure/prebid_server/stack_constants.py` file with default values of `CPU_TARGET_UTILIZATION_PCT = 66`, `MEMORY_TARGET_UTILIZATION_PCT = 50`, and `REQUESTS_PER_TARGET = 5000`.

The default minimum and maximum task counts are also set in the file as `TASK_MIN_CAPACITY = 2` and `TASK_COUNT_MAX = 300`, respectively.

AWS Fargate Spot compared to reserved instances

With Amazon ECS on AWS Fargate capacity providers, you can use both Fargate and Fargate Spot capacity with your Amazon ECS tasks. With Fargate Spot, you can run interruption tolerant Amazon ECS tasks at a rate that’s discounted compared to the Fargate price. When AWS needs the capacity back, tasks are interrupted with a 2-minute warning. When the Spot instances are terminated, ECS requests new Spot capacity, possibly from a different AZ to replace the reclaimed instances.

The default weights used for Fargate compared to Fargate Spot instances are defined through parameters in the solution source code `FARGATE_RESERVED_WEIGHT = 1` and `FARGATE_SPOT_WEIGHT = 1` respectively.

AWS Fargate container health check

Health check parameters that are specified in a container definition override any Docker health checks that exist in the container image. The following endpoint is used to check the health of the Prebid Server running containers: `HEALTH_ENDPOINT = "/status"`. The default interval and timeout parameters `HEALTH_CHECK_INTERVAL_SECS = 60` and `HEALTH_CHECK_TIMEOUT_SECS = 5` are used to set the time period in seconds between each health check run, and the time period in seconds to wait for a health check to succeed before it is considered a failure, respectively.

Prebid Server metrics

The Prebid Server collects various application metrics, which are submitted to configured backends. These metrics can be used for monitoring and optimizing the performance of the server. The metrics include information such as bid requests, bid responses, and other relevant statistics. For more detailed information on the specific metrics collected and how to integrate custom analytics adapters, refer to the official Prebid Server documentation and the metrics.md file in the GitHub repository for the Prebid Server Java implementation.

In this solution, metrics that are emitted by the Prebid Server containers are captured via log files. This is achieved by hooking the operational monitoring system to the Java file logger by modifying the original source file on the GitHub repository - MetricsConfiguration.java.

Operational and metrics Logs

Amazon EFS is mounted to the ECS cluster using NFS protocol and is shared among all container instances. Prebid Server writes two log files while running - one for `stdout/stderr` operational logging and one for auction metrics. They are both located on the EFS with a unique path based on the container instance to prevent collision. The operational log files generated by the Prebid Service container are located at `/mnt/efs/logs/<CONTAINER_ID>/prebid-server.log`. These files are rotated periodically, every day at midnight and placed in the archived folder located at `/mnt/efs/logs/<CONTAINER_ID>/archived/prebid-server.%d{yyyy-MM-dd}.%i.log.gz`.

If the generated log file exceeds a size limit of 100 MB, this rotation happens earlier. In a similar fashion, the metrics logs are located at `/mnt/efs/metrics/<CONTAINER_ID>/prebid-metrics.log` and archived at `/mnt/efs/metrics/<CONTAINER_ID>/archived/prebid-metrics.%d{yyyy-MM-dd_HH}.%i.log.gz`. These files are rotated at the top of each hour (period of one hour). The logging configurations are available through settings in the `deployment/ecr/prebid-server/prebid-logging.xml` file in the Prebid Server source code.

> **Note**
>
> If a container task receives a termination signal from ECS, for reasons such as scale-in or Spot interruptions, then the solution automatically compresses and transfers the current active log files to their respective archived folders to ensure no discontinuity or loss of log data. The transferred logs will be picked up along with any other log files in the `archived` directory when the DataSync task runs.

Transfer of log files

DataSync is configured to periodically (every hour on the half hour mark) to move the rotated and archived metric log files from EFS to an equivalent location in S3. For details, see the Logging section. This period is set through Cron schedule by `DATASYNC_METRICS_SCHEDULE = "cron(30 * * * ? *)"` and `DATASYNC_LOGS_SCHEDULE = "cron(30 * * * ? *)"`. These parameters are available through the `source/infrastructure/prebid_server/stack_constants.py` file in the solution source code.

After each file is copied to S3 and verified, it is removed from the EFS file system through a Lambda function. Only actively written log files are retained on the EFS.

Metrics processing

An AWS Glue job is configured to perform ETL operation on the metrics log data present in the S3 bucket containing the raw logs. The ETL operation structures the metric data present in the log files into a single database with several tables. For details, see Glue table schemas in the Querying metrics with Athena section. The processed data is written into the `MetricsEtl` S3 bucket, which contains the metric log data transformed and partitioned through ETL in the previous steps. The data is made available via AWS AWS Glue Data Catalog, a persistent technical metadata store for analytics users to discover and prepare the data. Also, you can immediately search and query cataloged data using Amazon Athena, Amazon EMR, and Amazon Redshift Spectrum.

Logging

The log files generated by this solution are stored in four separate buckets depending on their context. Logs containing operational data, such as CloudFront access logs, are stored in one bucket, while logs containing business intelligence, such as Prebid Server performance metrics, are stored in another bucket. This organization is intended to facilitate data pipelines for context-specific log analysis.

The following S3 buckets are used for log storage.

CloudFront access logs bucket

Contains detailed information about every request received by the CloudFront distribution.

ALB access logs bucket

Contains detailed information about every request received by the Application Load Balancer (ALB). Elastic Load Balancing provides access logs that capture detailed information about requests sent to your load balancer. Each log contains information such as the time the request was received, the client's IP address, latencies, request paths, and server responses. You can use these access logs to analyze traffic patterns and troubleshoot issues. For more information, see Access logs for your Application Load Balancer in Elastic Load Balancing.

DataSync logs bucket

- Contains `stdout` and `stderr` log streams from the Prebid Server processes running in the ECS cluster.
- Each log stream is saved in a unique directory named according to its container identifier in ECS.

  To learn more about these logs, see the Prebid Server operation and metrics logs section.

DataSync metrics bucket

- Contains metric streams for Prebid Server transactions running in the ECS cluster.
- Each metric stream is saved in a unique directory named according to its container identifier in ECS.

  To learn more about these logs, see the Prebid Server operation and metrics logs section.

Prebid Server containers in Amazon ECS send log information to CloudWatch Logs. These logs can be found within the `AWS::Logs::LogGroup` resource shown in the CloudFormation stack. The log group includes output from the commands specified in the Dockerfile that is used to assemble Prebid Server containers. Since the Prebid Server process redirects both `stdout` and `stderr` to EFS, the CloudWatch logs will be empty unless ECS encountered a problem running the Prebid Server process.

Prebid Server operation and metrics logs

This solution uses EFS to store logs from every Prebid Server task running in ECS. Each task writes runtime logs to `/mnt/efs/logs/<CONTAINER_ID>` and metrics logs to `/mnt/efs/metrics/<CONTAINER_ID>` . The container identifier in the path ensures that each Prebid Server instance stores logs without interfering with the logs from other instances.

The logging policy for the prebid process is defined in the prebid-logging.xml file. It includes the following customizations:

- Info level logging of `stdout/stderr` output.
- Rolling policy to archive logs when they exceed 100 MB or at a predetermined frequency of once per hour.
- Asynchronous logging to minimize the computational overhead of logging. For more information, see Benchmarking synchronous and asynchronous logging.

The log files are archived periodically (every 60 minutes) and moved to the corresponding archive folders `/mnt/efs/logs/<CONTAINER_ID>/archived` and `/mnt/efs/metrics/<CONTAINER_ID>/archived`.

DataSync moves the archived log files from EFS to S3 every 60 minutes. Runtime logs are saved to `s3://<STACK_NAME>-datasynclogsbucket-<RANDOM_STRING>/<CONTAINER_ID>/archived`. Metrics logs are saved to `s3://<STACK_NAME>-datasyncmetricsbucket-<RANDOM_STRING>/<CONTAINER_ID>/archived`.

If you need to access the Prebid Server logs before they are copied by AWS DataSync, then mount the EFS filesystem directly using the procedure described in Accessing Prebid Server logs from EFS.

Plan your deployment

This section describes the cost, configurations for ECS Auto Scaling, security, Region, and quota considerations for planning your deployment.

Supported AWS Regions

Prebid Server on AWS is supported in the following AWS Regions:

Region name	
US East (Ohio)	Asia Pacific (Tokyo)
US East (N. Virginia)	Canada (Central)
US West (Northern California)	Europe (Frankfurt)
US West (Oregon)	Europe (Ireland)
Asia Pacific (Mumbai)	Europe (London)
Asia Pacific (Osaka)	Europe (Paris)
Asia Pacific (Seoul)	Europe (Stockholm)
Asia Pacific (Singapore)	South America (São Paulo)
Asia Pacific (Sydney)	

Cost

You are responsible for the cost of the AWS services used while running this solution. As of this revision, the cost for running this solution with the default settings with no incoming bidding traffic to the solution in the US East (N. Virginia) Region is approximately **$237.50 per month** .

We recommend creating a budget through AWS Cost Explorer to help manage costs. Prices are subject to change. For full details, see the pricing webpage for each AWS service used in this solution.

Note

If you want to opt out of using CloudFront and AWS WAF and directly send requests to the ALB, see Opting out of using CloudFront and AWS WAF.

Cost tables

The total cost of this solution includes two parts:

- Cost for deploying the solution without incoming bidding traffic
- Cost incurred from the traffic flowing through the solution

The following assumptions apply for calculating the cost:

- The cost breakdown per month includes deploying this solution with the default parameters in the US East (N. Virginia) Region.
- The Fargate to Fargate Spot pricing ratio is 1:1. With Fargate Spot instances, customers can run interrupt-tolerant Amazon ECS tasks on spare capacity at up to a 70% discount off of the regular Fargate instance price. For more information, see AWS Fargate Pricing.
- Each incoming HTTP request to the solution is of a fixed size (0.5 KB), and three bidders participate in a single auction request.

Sample cost table 1

No incoming bidding traffic to the solution, two Amazon ECS tasks

AWS service	Dimensions	Monthly cost [USD]
Amazon ECS	Operating system (Linux), CPU architecture (x86), Average duration (30 days), Number of tasks or pods (2 per month), Amount of memory allocated (4 GB), Amount of ephemeral storage	$54.50

AWS service	Dimensions	Monthly cost [USD]
	allocated for Amazon ECS (20 GB)	
AWS WAF	Number of Web Access Control Lists (Web ACLs) utilized (1 per month), Number of Managed Rule Groups per Web ACL (6 per month)	$15.00
Elastic Load Balancing	Number of Application Load Balancers (1)	$17.00
Amazon EC2 - other	Number of NAT gateways (2) DT inbound: Not selected (0 TB per month), DT outbound: Internet (<50 GB per month), DT Intra-Region: (0 TB per month)	$69.00
Amazon EFS	Desired storage capacity (1 TB per month), Infrequent access requests (<2 GB per month)	$25.00
Amazon S3	S3 Standard storage	$4.00
Amazon CloudWatch	Number of Standard Resolution Alarm Metrics (20), Standard logs: Data ingested (<20 GB)	$10.00
Other servicess	Amazon CloudFront, AWS CloudTrail AWS DataSync, IAM, AWS Glue, AWS KMS, AWS Lambda, and Amazon VPC	$47.00

AWS service	Dimensions	Monthly cost [USD]
	Total:	**$241.50**

Sample cost table 2

1,500 incoming requests per second (appoximately 1.3B × 3 = 3.7B impressions per month), 10 tasks total

AWS service	Dimensions	Monthly Cost [USD]
Amazon CloudFront	Data transfer out to origin (1.5 TB per month), Data transfer out to internet (0.75 TB per month), Number of requests (HTTPS) (1.3 billion per month)	$967.50
Amazon ECS	Operating system (Linux), CPU architecture (x86), Average duration (30 days), Number of tasks or pods (10 per month), Amount of memory allocated (4 GB), Amount of ephemeral storage allocated for Amazon ECS (20 GB)	$276.00
AWS WAF	Number of Web Access Control Lists (Web ACLs) utilized (1 per month), Number of Managed Rule Groups per Web ACL (6 per month)	$211.00
Elastic Load Balancing	Number of Application Load Balancers (1)	$133.00

AWS service	Dimensions	Monthly Cost [USD]
Amazon EC2 - other	Number of NAT Gateways (2) DT inbound: Not selected (0 TB per month), DT outbound: Internet (1.5 TB per month), DT Intra-Region: (0 TB per month), Data transfer cost (13.5)	$273.00
Amazon EFS	Desired storage capacity (1 TB per month), Infrequent access requests (2 GB per month)	$96.00
Amazon S3	S3 Standard storage (2.5 TB per month)	$59.00
Amazon CloudWatch	Number of Standard Resolution Alarm Metrics (20), Standard logs: Data ingested (41 GB)	$23.00
AWS Glue	Data processing unit-hour for AWS Glue ETL job, approx. 200 DPU-hour	$92.00
	Total:	**$2,130.50**

Sample cost table 3

9,000 incoming requests per second (7.8B × 3 = 23B impressions per month), 60 Amazon ECS tasks

AWS service	Dimensions	Monthly cost [USD]
Amazon CloudFront	Data transfer out to origin (30 TB per month), Data transfer out to internet (5.75	$6,256.00

AWS service	Dimensions	Monthly cost [USD]
	TB per month), Number of requests (HTTPS) (7.8 billion per month)	
Amazon ECS	Operating system (Linux), CPU architecture (x86), Average duration (30 days), Number of tasks or pods (60 per month), Amount of memory allocated (4 GB), Amount of ephemeral storage allocated for Amazon ECS (20 GB)	$1,660.00
AWS WAF	Number of Web Access Control Lists (Web ACLs) utilized (1 per month), Number of Managed Rule Groups per Web ACL (6 per month)	$1,717.00
Elastic Load Balancing	Number of Application Load Balancers (1)	$717.00
Amazon EC2 - other	Number of NAT Gateways (2) DT inbound: Not selected (0 TB per month), DT outbound: Internet (1.2 TB per month), DT Intra-Region: (0 TB per month), Data transfer cost (13.5)	$748.00
Amazon EFS	Desired Storage Capacity (1 TB per month), Infrequen t access requests (5 GB per month)	$96.00

AWS service	Dimensions	Monthly cost [USD]
Amazon S3	S3 Standard storage (5 TB per month)	$118.00
Amazon CloudWatch	Number of Standard Resolution Alarm Metrics (20), Standard logs: Data ingested (41 GB)	$23.00
AWS Glue	Data processing unit-hour for AWS Glue ETL job, approx. 200 DPU-hour	$92.00
	Total:	**$11,427.00**

Configurations for Elastic Container Service (ECS) Auto Scaling

The recommended configurations for the deployed solution's automatic scaling are dependent on the approximate maximum requests per second (RPS) and maximum number of users the solution is expected to support.

In this context, RPS means HTTP or HTTPS requests per second. A single request can contain multiple bid requests that can result in multiple bid responses inside the HTTP response. The request and response might both contain a payload. The average response time refers to the amount of time it takes to receive winning bids, measured in seconds, and the timer starts when the requests for advertisement bids are sent out and stops when the winning bids are received.

The recommendations in this section were determined via load testing with Distributed Load Testing on AWS. In the load tests, 10,000 users with 16.7 new users being added per second were spawned across `us-east-1`, `us-west-1`, `us-east-2`, and `us-west-2` Regions to generate traffic to the Prebid server cluster.

In the context of load testing, a user continuously makes an auction request to the auction API. 80% of the total RPS are auction API requests. The user infrequently sends requests to the non-auction APIs. This includes information and status check requests. The approximate average payload sizes for an API request and response is 123 KB and 331 KB respectively.

The statistics in the tables below were calculated by the data collected from `us-east-1`, `us-west-1`, `us-east-2`, and `us-west-2` Regions.

Static cluster size configurations

The following table lists the recommended static cluster sizes and their associated maximum stable RPS limits, average response time, and success rate if ECS Auto Scaling is turned off.

ECS number of tasks with no Auto Scaling	Transactions per second	Average response time in seconds	Success rate
1	800.79	9.56630	87.70%
10	4145.95	1.84996	97.38%
25	11074.86	0.69569	98.93%
50	75912	0.35765	99.60%
100	75411	0.17981	99.89%
200	64621.02	0.13120	99.86%
400	128793.61	0.07452	99.97%

Significant latency and failed requests were observed when the traffic was exceeded for each number of tasks tested. Further increases to the number of tasks were able to handle the 10,000-user test load with better success rate, average response time, and RPS.

Auto Scaling cluster configurations

Turning on Auto Scaling in ECS increases the performance of the solution's maximum RPS. The following recommended ECS Auto Scaling policies and parameters were used in the load tests.

Parameters:

- Minimum number of tasks: 10
- Maximum number of tasks: 100

Policies:

- `ALBRequestCountPerTarget`
  - Target value: 5000
  - Scale-out cooldown period: 300
  - Scale-in cooldown period: 300
- `ECSServiceAverageCPUUtilization`
  - Target value: 66%
  - Scale-out cooldown period: 300
  - Scale-in cooldown period: 300
- `ECSServiceAverageMemoryUtilization `
  - Target value: 50%
  - Scale-out cooldown period: 300
  - Scale-in cooldown period: 300

The following table lists the auto-scaling policies and their associated maximum stable RPS limits, average response time, and success rate.

Auto-scaling policies	Min number of tasks scaled	Max number of tasks scaled	Transactions per second	Average response time in seconds	Success rate
ALBRequestCountPerTarget (ALB)	10	100	17367.6	0.44486	99.51%
ECSServiceAverageCPUUtilization (CPU)	10	13	4708.65	1.85956	96.41%
ECSServiceAverageMemoryUtil	10	12	5820.56	1.31274	98.59%

Auto-scaling policies	Min number of tasks scaled	Max number of tasks scaled	Transactions per second	Average response time in seconds	Success rate
ization (Mem)					
ALB & CPU	10	100	14948.84	0.51504	99.48%
ALB & Mem	10	100	15208.86	0.50105	99.50%
CPU & Mem	10	13	4747.65	1.60875	97.08%
ALB & CPU & Mem	10	100	16211.21	0.49361	99.42%

The `ALBRequestCountPerTarget` policy is the most important auto-scaling policy and plays the biggest influence on the performance. However, we recommend that you use all three of the Auto Scaling policies above. Removing them will decrease the maximum RPS and increase response time because then the containers are more prone to becoming overloaded. The policies also make the deployed solution more resilient to cases where there is a burst of users.

The maximum number of tasks and minimum number of tasks can be adjusted depending on the solution's usage. We recommend to at least have 50 tasks and have Auto Scaling turned on for the deployed solution to reduce response times and the chance of errors occurring.

Fargate Spot instances ratio configurations

We recommend that you keep the solution's default 50:50 ratio of the Fargate Spot instances to Fargate instances at least. This is because during testing, the Fargate instances were found to help the system scale and react more quickly to user traffic and support higher RPS more quickly with higher success rate.

The following table lists the Fargate Spot instances ratio and their associated maximum stable RPS limits, average response time, and success rate.

Fargate: Fargate Spot	Transactions per second	Average response time in seconds	Success rate
50:50	17789.75	0.43171	99.60%
100:0	134244.83	0.07305	100%

Example cluster size, Auto-scaling policy, and Fargate Spot instances ratio configurations

You can use the following specifications for Prebid Server, based upon the testing conducted in this document.

Parameters:

- Minimum number of tasks: 50
- Maximum number of tasks: 400

Policies:

- `ALBRequestCountPerTarget`
  - Target value: 5000
  - Scale-out cooldown period: 300
  - Scale-in cooldown period: 300
- `ECSServiceAverageCPUUtilization`
  - Target value: 66%
  - Scale-out cooldown period: 300
  - Scale-in cooldown period: 300
- `ECSServiceAverageMemoryUtilization`
  - Target value: 50%
  - Scale-out cooldown period: 300
  - Scale-in cooldown period: 300

Fargate Spot instances ratio:

- Fargate instances: 80
- Fargate Spot instances: 20

The metrics achieved in testing with the above configurations are in the following table.

Results from recommended configurations	
Maximum transaction per second	190881.19
Average response time in seconds	0.05533
Success rate	100%

Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared responsibility model reduces your operational burden because AWS operates, manages, and controls the components including the host operating system, the virtualization layer, and the physical security of the facilities in which the services operate. For more information about AWS security, visit AWS Cloud Security.

IAM roles

AWS Identity and Access Management (IAM) roles allow customers to assign granular access policies and permissions to services and users on the AWS Cloud. This solution creates IAM roles and policies with minimal permission that grant access to the solution's resources.

Amazon CloudFront

This solution deploys an Amazon CloudFront distribution and uses the default CloudFront domain name and SSL certificate. The default CloudFront SSL certificate only supports TLSv1. To use a later TLS version (TLS1.2 and above), use your own domain name and custom SSL certificate. For more information, refer to Using alternate domain names and HTTPS in the *Amazon CloudFront Developer Guide*.

The Amazon CloudFront distribution is the unified network entry point. It helps to reduce latency by delivering data through globally dispersed points of presence (PoPs) with automated network mapping and intelligent routing. The inbound traffic to the CloudFront can be either HTTP or

HTTPS for compatibility with various clients. As the Prebid Server hosted by ECS only supports HTTP, HTTPS proxy design is used in this solution to improve security. The CloudFront distribution acts as a TLS proxy, where APIs can be delivered over HTTPS using the latest version Transport Layer Security (TLSv1.3) to encrypt and secure communication between viewer clients and the CloudFront.

Application Load Balancer (ALB)

ALB distributes incoming request traffic for Prebid Server through the cluster of containers. ALB provides a single entry point into the cluster, and it is the primary origin for the CloudFront distribution. CloudFront and the ALB use a shared secret header to prevent external traffic from bypassing the CloudFront distribution and accessing ALB directly. For more information, see Restricting access to Application Load Balancers in the *Amazon CloudFront Developer Guide*.

Amazon VPC

Amazon VPC is configured with multiple subnets, routes, security groups, and NAT gateways. Security groups permit traffic to and from the subnets. The VPC contains the network interfaces for the Prebid Server cluster nodes. ALB has an interface in each private subnet of the VPC. Each container instance (or node) has an interface in its private subnet of the VPC. The actual number of interfaces varies based on the number of containers running. The VPC is configured for private IP addresses only, and container networks configured within the VPC use the NAT gateway as a default route to the internet for communication.

AWS Fargate

This solution uses Amazon ECS to containerize the Prebid Server. The container runs the open source Prebid Server and is hosted by the Elastic Container Repository for AWS Solutions. A custom build is applied to the open source project's default container in configuration settings for areas including file output and bidder adapter. To see how the ECS containers are constructed, refer to the Dockerfile. If you need to access these containers while they are running in Fargate, see Using Amazon ECS Exec to access your containers on AWS Fargate and Amazon EC2 on the *AWS Blog*.

Security groups

This solution creates an Amazon EC2 security group within an Amazon VPC and associates it with ALB to act as a virtual firewall for the EC2 instances to control incoming and outgoing traffic. A rule with prefix list is added to the security group that allows ingress only from the CloudFront distribution.

AWS WAF

This solution deploys AWS WAF and Shield Standard as a protection mechanism from DDoS attacks against the Prebid Server cluster. One or more managed rule groups can be activated in AWS WAF by default after extended testing including rules in the Baseline Rule Group and the IP Reputation Rule Group. AWS WAF allows the securement of web applications' API from attacks before reaching to the servers. The customer has the option to activate, purchase, or use existing rule subscriptions, or add regular expression or CIDR matching rules as needed.

Customer managed AWS KMS keys

AWS Key Management Service (KMS) lets customers create and manage cryptographic keys to activate server-side encryption. This solution creates six KMS keys and uses them in the S3 buckets storing artifacts, CloudFront access log data, CloudTrail events, DataSync log and metric files, and metadata of AWS AWS Glue Data Catalog to secure data at rest.

Audit trails

CloudTrail is configured in this solution for auditing API calls against resources and used for problem analysis and remediation. CloudWatch alarms are used by compute resources for software failures. All compute resources send logging output to CloudWatch logs. CloudFront standard logs are configured to create log files that contain information about the user requests initiated to the solution's CloudFront distribution.

Quotas

Service quotas, also referred to as limits, are the maximum number of service resources or operations for your AWS account.

Quotas for AWS services in this solution

Make sure you have sufficient quota for each of the services implemented in this solution. For more information, see AWS service quotas.

Use the following links to go to the page for that service. To view the service quotas for all AWS services in the documentation without switching pages, view the information in the Service endpoints and quotas page in the PDF instead.

AWS CloudFormation quotas

Your AWS account has AWS CloudFormation quotas that you should be aware of when deploying this solution. By understanding these quotas, you can avoid limitation errors that would prevent you from deploying this solution successfully. For more information, see AWS CloudFormation quotas in the in the *AWS CloudFormation User's Guide*.

Deploy the solution

This solution uses AWS CDK and stacks to automate its deployment. The CDK code specifies the AWS resources included in this solution and their properties. The CDK stack provisions the resources that are described in the template.

> ⚠ **Important**
>
> This solution includes an option to send anonymized operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. AWS owns the data gathered through this survey. Data collection is subject to the AWS Privacy Notice.
> To opt out of this feature, modify the CDK code before deploying. For more information, see the Anonymized data collection section of this guide.

Prerequisites

You need an AWS account with permissions to deploy CDK stack and all the resources defined within the stack for this solution. The `AdministratorAccess` IAM policy, which provides full access to AWS services and resources is sufficient to deploy this solution.

Deployment process overview

Deploy the solution guidance using the AWS CDK stack available in the Github repository.

Before you launch the solution, review the cost, architecture, network security, and other considerations discussed earlier in this guide.

Follow the steps in the section "Deployment" in the Git repo readme. The default configuration deploys the core and supporting services found in the AWS services in this solution section, but you can customize the CDK stack to meet your specific needs.

You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive a CREATE_COMPLETE status in approximately 10 minutes.

Time to deploy: Approximately 10 minutes

CDK Stack Parameters

The following table lists the parameters for the stack.

Parameter	Default	Description
PrebidServerContainerImage	`public.ecr.aws/aws-solutions/prebid-server:{solution version}`	The fully qualified name of the Prebid Server container image to deploy. Set the env variable OVERRIDE_ECR_REGISTRY to point the deployment to a custom container registry.
deploy_cloudfront_and_waf_param	`Yes`	Specifies whether to deploy a CloudFront distribution. If you set it to `Yes`, the solution creates and configures a CloudFront distribution to serve your content. If you set it to No, provide your own SSL certificate.
ssl_certificate_param	*Optional*	The ARN of an SSL certificate in AWS Certificate Manager associated with a domain name. This field is only required if **deploy_cloudfront_and_waf_param** is set to No.
ECSTaskMinCapacity	*2*	The minimum number of running tasks that Amazon ECS will maintain for the Prebid Server service when ECS is autoscaling based on

Parameter	Default	Description
		traffic, CPU, and memory utilization.
ECSTaskMaxCapacity	*300*	The maximum number of running tasks that Amazon ECS will maintain for the Prebid Server service when ECS is autoscaling based on traffic, CPU, and memory utilization.
RequestsPerTargetThreshold	*5000*	The number of requests per target to trigger scaling up the Prebid Server ECS service.
SpotInstanceWeight	*1*	Spot instance weight configuration (On-demand weight fixed at 1).

Note

CloudFront distribution is the recommended configuration for this solution because this configuration provides enhanced security and performance. Refer to the Opt out of using CloudFront and AWS WAF section for more information about the benefits of using CloudFront and AWS WAF together, and steps to disable CloudFront and AWS WAF if desired.

Configure the solution

This section provides instructions for configuring Guidance for Deploying a Prebid Server on AWS.

Opting out of using CloudFront and AWS WAF

By default, this solution deploys CloudFront and AWS WAF.

Benefits of using CloudFront and AWS WAF

CloudFront helps reduce latency by delivering data through globally dispersed Points of Presence (PoPs) with automated network mapping and intelligent routing. It cuts costs with consolidated requests, customizable pricing options, and zero fees for data transfer out from AWS origins. CloudFront can cache objects and serve them directly to users (viewers), reducing the load on your Application Load Balancer.

The Application Load Balancer is configured to forward requests that contain a custom secret header value to enhance security. CloudFront automatically adds this custom HTTP header to the requests. This secret value is unique and is generated when you deploy the stack. For added protection, the Application Load Balancer security group is configured with the CloudFront managed prefix list. This contains the IP address ranges of all CloudFront globally distributed origin-facing servers. The CloudFront managed prefix list allows inbound traffic to your origin only from CloudFront origin-facing servers, preventing any non CloudFront traffic from reaching your origin.

AWS WAF provides additional security by preventing distributed denial of service (DDoS) and helps more easily monitor, block, or rate-limit common and pervasive bots. It improves web traffic visibility with granular control over how metrics are emitted.

How to opt out

For users who decide not to use CloudFront and AWS WAF with this solution, follow these steps:

1. Obtain an SSL server certificate for your domain by requesting or importing it in AWS Certificate Manager (ACM) in the same Region where you plan to launch your stack.
2. Validate the ACM certificate request by adding the required DNS CNAME record.

3. Before doing cdk deploy set the AWS CDK stack parameter **deploy_cloudfront_and_waf_param** in the file stack_cfn_parameters.py to No.
4. Enter the SSL certificate ARN in AWS CDK stack parameter **ssl_certificate_param** .
5. After you deploy the stack, access the ALB DNS name from the AWS CloudFormation console, on the **Outputs** tab, from the value for **PrebidALBDNSName**.
6. Update your domain's public hosted zone by creating a CNAME record that points to the ALB's Fully Qualified Domain Name (FQDN) obtained in step 5.

Managing configuration files for the solution

This section provides detailed instructions for updating configuration files for the Prebid Server and restarting the cluster. It also covers file descriptions, recovery from configuration mistakes, and guidance on using the S3 bucket's versioning feature for rollback.

Finding the S3 bucket for prebid configuration files

The S3 bucket used for Prebid Server configuration files is created automatically by the stack. To locate the bucket where the configuration files are stored, follow these steps:

1. **Access the CloudFormation stack outputs** - Sign in to the AWS CloudFormation console.
2. **Select the Prebid Server stack** - From the list of stacks, select the stack that is responsible for creating your Prebid Server environment.
3. **View the outputs** - After selecting the stack, choose the **Outputs** tab.
4. **Locate the S3 bucket link** - Find the **Output** entry that starts with `ContainerImagePrebidSolutionConfigBucket`. This output contains a direct link to the S3 bucket where the configuration files are stored.
5. **Navigate to the S3 bucket** - Choose the link in the output to navigate to the root of the S3 bucket. From there, you see the folder structure containing the `default/` and `current/` configuration files.

After you locate the bucket, you can download, modify, and manage the Prebid Server configuration files as described in the following sections.

This S3 bucket is retained after the stack is deleted. To delete the bucket, see Deleting the Amazon S3 buckets

Making configuration changes

When working with the Prebid Server, maintain separation between default and custom configurations. Make updates to the configuration files in the `/prebid-server/current/` folder in your S3 bucket, and never in the `/prebid-server/default/` folder. After you update the files, redeploy your cluster to apply the changes.

Important folders:

- `default/` - Stores baseline configuration files for Prebid Server, which should not be modified.
- `current/` - Used for custom configuration files that override the defaults.

Updating configuration files and restarting the cluster

1. **Identify the configuration files to modify**
   1. Check the `/prebid-server/default/` folder in the S3 bucket for the baseline configuration files.
   2. Download the relevant file from the `/prebid-server/default/` folder to your workstation. You cannot modify the files directly in the S3 bucket.
2. **Edit the configuration files**
   1. After you download the file to your workstation, make the required changes.
   2. Upload the modified file back to the `/prebid-server/current/` folder in your S3 bucket.

      Common configuration files include:

      - `prebid-config.yaml` - Contains the main configuration settings for the Prebid Server instance.
      - `prebid-logging.xml` - Manages logging levels and output formats for Prebid Server logs.
      - `entrypoint.sh` - Custom script that initializes and launches the Prebid Server containers.

   For detailed information about how to configure Prebid Server, refer to the Prebid Server Java Documentation.
3. **Document custom scripts**

   If you need to modify or add a script such as `entrypoint.sh` (which is custom), ensure that the following details are documented:

   a. **Purpose** - What the script is used for (for example, starting services, managing environment variables).

   b. **Usage** - Specific configurations, dependencies, or parameters used.

   c. **Modification** - How to safely edit the script.

   Example: The `entrypoint.sh` script ensures that the necessary environment variables are set before starting the Prebid Server. It can be updated to handle different runtime configurations.

4. **Test your changes in a test stack**

   Before applying changes to the production environment, test the configuration updates. Deploy a test stack in an AWS account to ensure that the changes work as expected, avoiding disruption to the production environment.

5. **Deploy the custom configuration**

   After making and testing your changes, complete the following steps:

   a. Upload the updated files to the `/prebid-server/current/` folder in the S3 bucket.

   b. Force a redeployment of your ECS cluster by following Amazon ECS documentation for initiating a redeployment.

6. **Verify the deployment**

   After you complete the redeployment, verify that the new configuration has been applied and that the Prebid Server is functioning as expected. Check the logs and monitor key metrics to ensure that no errors are occurring.

Description of important files

1. `prebid-config.yaml`

   Contains the core configuration for running Prebid Server. This file manages settings such as:

   - Bidder adapters
   - Server ports
   - Caching rules

   Refer to the Prebid Server Configuration Guide for detailed usage instructions.

2. `prebid-logging.xml`

Defines logging levels and output destinations for Prebid Server logs. Adjustments to this file control the verbosity and detail of logging, critical for debugging and auditing purposes.

Example modification: Change the logging level from `INFO` to `DEBUG` for troubleshooting.

Documentation: Prebid Server Logging.

3. `entrypoint.sh`

   This is the script responsible for initializing the Prebid Server container. It sets up environment variables, runs prerequisite commands, and starts the Prebid Server.

   Document custom steps or configurations made to this script.

4. **Custom files**

   Document the custom files added to the `/current/` folder in terms of their purpose, usage, and changes that can be made.

Recovering from a mistake in the config files

Mistakes in configuration files can lead to failed deployments or runtime errors. Because S3 bucket versioning is enabled, you can revert to previous configurations. See How S3 Versioning works for more information.

1. **Identify the issue**

   Review the logs or errors generated during deployment. You can access logs through several methods:

   - Navigate to the **Logs** tab under the Prebid Amazon ECS service in the AWS Management Console to review consolidated output from all containers.
   - Navigate to a specific **Task** in Amazon ECS, and view the **Logs** tab for output from that specific container.
   - Check **CloudWatch Logs** in the log group named `<STACK-NAME>-PrebidContainerLogGroup[.red]<ID>` . Use Log Insights or Live Tail to observe and search the logs.

     For more detailed instructions on how to access Amazon ECS and CloudWatch Logs, refer to the Amazon ECS Logs documentation.

2. **Check S3 version history**

   Navigate to your S3 bucket and use versioning to locate the previous, stable version of the configuration file:

   a. Select the file from the `/current/` folder and enable the **Show versions** switch.

   b. Locate the correct version of the file before the mistake was introduced.

3. **Restore the previous version**

   After you've identified the correct file version, restore it by selecting it and replacing the current file. You can also remove the problematic file from the `/current/` folder to allow the original file in the `/default/` folder to take precedence.

4. **Redeploy the cluster**

   After restoring the stable version, follow the redeployment steps to update your Amazon ECS cluster with the corrected configuration.

Best practices for recovery

- **Backup** - Always create a backup of your current configuration before making any changes.
- **Testing** - Test configurations in a test stack before applying changes to production.
- **Document** - Keep detailed notes on the changes made, including the rationale behind the modifications, to simplify troubleshooting and rollback if needed.

By maintaining a clear separation between default and custom configurations and using S3 bucket versioning, you can safely and effectively update your Prebid Server setup. Always test thoroughly and document changes to ensure smooth deployments and easy recovery from any configuration issues.

Tune the solution

CloudFront distribution domain name

The CloudFront distribution in this solution uses the default CloudFront domain name (`xxxxxxxxxxxxxx.cloudfront.net`) and certificate (`\*.cloudfront.net`). To use a custom domain name and HTTPS between viewers and CloudFront, see Configuring alternate domain names and HTTPS in the *Amazon CloudFront Developer Guide* after stack deployment.

Guidance on how to implement/enable TLS between CloudFront and ALB

You can configure your CloudFront distribution to always use HTTPS when sending requests to your Application Load Balancer. Remember, this only works if you keep the custom header name and value secret. Using HTTPS can help prevent an eavesdropper from discovering the header name and value. We also recommend rotating the header name and value periodically.

Use HTTPS for origin requests

To configure CloudFront to use HTTPS for origin requests, set the origin protocol policy setting to HTTPS Only. This setting is available in the CloudFront console, AWS CloudFormation, and the CloudFront API. For more information, see Protocol (custom origins only) in the *Amazon CloudFront Developer Guide*.

The following also applies when you configure CloudFront to use HTTPS for origin requests:

- You must configure CloudFront to forward the host header to the origin with the managed origin request policy. For details, see AllViewer managed origin request policy.
- Make sure that your Application Load Balancer has an HTTPS listener. For more information, see Create an HTTPS listener for your Application Load Balancer in *Elastic Load Balancing*. Using an HTTPS listener requires you to have an SSL/TLS certificate that matches the domain name that's routed to your ALB.
- SSL/TLS certificates for CloudFront can only be requested (or imported) in the us-east-1 Region in AWS Certificate Manager (ACM). Because CloudFront is a global service, it automatically distributes the certificate from the us-east-1 Region to all Regions associated with your CloudFront distribution.

  - For example, if you have an ALB in the ap-southeast-2 Region, you must configure SSL/TLS certificates in both the ap-southeast-2 Region (for using HTTPS between CloudFront and ALB origin) and the us-east-1 Region (for using HTTPS between viewers and CloudFront). Both certificates should match the domain name that is routed to your Application Load Balancer. For more information, see AWS Region for AWS Certificate Manager in the *Amazon CloudFront Developer Guide*.
- If the end users of your web application can use HTTPS, you can also configure CloudFront to prefer (or even require) HTTPS connections from the end users. To do this, use the **Viewer Protocol Policy** setting. You can set it to redirect end users from HTTP to HTTPS, or to reject requests that use HTTP. This setting is available in the CloudFront console, AWS CloudFormation, and the CloudFront API. For more information, see Viewer protocol policy in the *Amazon CloudFront Developer Guide*.

Firewall rules

This solution uses AWS WAF as a protection mechanism from DDoS attacks against the Prebid Server cluster. The web access control list (web ACL) gives customers fine-grained control over the web requests that the Amazon CloudFront distribution responds to. This solution adds AWS Managed Rules in the web ACL but users must add more web ACLs according to their own environment, such as a referrer rule to reject requests when referrer headers don't match supported websites. For more information, see AWS WAF rules in the *AWS WAF, Firewall Manager, and AWS Shield Advanced Developer Guide*.

To add new rules, open the web ACL corresponding to the `PrebidWAFWebACL` resource listed in the CloudFormation stack outputs. For more information, see Web access control lists (web ACLs) in the *AWS WAF, Firewall Manager, and AWS Shield Advanced Developer Guide*.

Container tuning

The performance and cost of running this solution are dominated by utilization of the ECS cluster. The largest user-configurable factors that users can customize to affect performance, cost, and efficiency are ECS Service Auto Scaling, min and max container size, and Fargate / Fargate Spot capacity providers. For the effect of ECS service configuration on performance and cost, refer to the Cost and Configurations for Elastic Container Service (ECS) Auto Scaling sections in this guide.

This solution has Service Auto Scaling in place to add or remove service tasks based on metric and target value. For more information about automatic scaling in ECS, refer to Automatically scale your Amazon ECS service and Scale your Amazon ECS service using a target metric value in the *Amazon ECS Developer Guide*.

This solution uses a weighted combination of Fargate and Fargate Spot instances with defaults to 50/50. For information about Fargate Spot tasks. see AWS Fargate capacity providers in the *Amazon ECS Developer Guide* and Deep dive into Fargate Spot to run your ECS Tasks for up to 70% less on the *AWS Blog*. You can perform the procedures in Updating a service using the console in the *Amazon ECS Developer Guide*, to edit Service Auto Scaling policies, Fargate Spot instances ratio. To adjust how much CPU and memory to use with each task, see Updating the task definition using the console in the *Amazon ECS Developer Guide*. Also, see Amazon ECS task definitions.

Monitor the solution

This solution defines several CloudWatch alarms to monitor its health and performance. Each alarm has a fixed threshold or an anomaly-based threshold that causes state to transit between OK and ALARM. Review the full list of alarms with details in the Amazon CloudWatch Alarms section. You can view the alarms in the Amazon CloudWatch console and configure alarm notifications as needed after stack deployment. For general information about CloudWatch alarms and alarm actions, see Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*.

All compute resources in this solution send log information to CloudWatch Logs. These logs are available in CloudWatch for querying and searching in real time. This solution uses CloudTrail to track S3 and Lambda API activities with data events. The CloudTrail log files are stored in an S3 bucket, and the event history can be viewed in CloudTrail.

CloudFront access logs are configured to create log files containing information about user requests initiated to the solution's CloudFront distribution. To analyze the access logs in Amazon Athena, see Querying Amazon CloudFront logs in the *Amazon Athena User Guide*.

Prebid Server tasks running in the ECS cluster output the runtime logs into the mounted Amazon Elastic File System which is shared across containers, and store the log data long-term in S3. For more information about Prebid Server logs, see the Logging section.

You can monitor how the ECS resources in this solution are performing using the cluster and service metrics that are available in the Amazon ECS console. To view these metrics, follow the steps in Viewing Amazon ECS metrics in the *Amazon ECS Developer Guide*.

This solution associates a web ACL with the CloudFront distribution to prevent DDoS attacks against the Prebid Server cluster. Users can access near real-time summaries of the traffic that the web ACL evaluates in the web ACL's **Traffic overview** tab on the AWS WAF console. For details, see Web ACL traffic overview dashboards in the *AWS WAF, AWS Firewall Manager, and AWS Shield Advanced Developer Guide*.

Traffic monitoring and troubleshooting

This section provides traffic monitoring and troubleshooting instructions for deploying and using the solution. If this information don't help address your issue, Contact Support provides instructions for opening an AWS Support case for this solution.

Amazon CloudWatch alarms

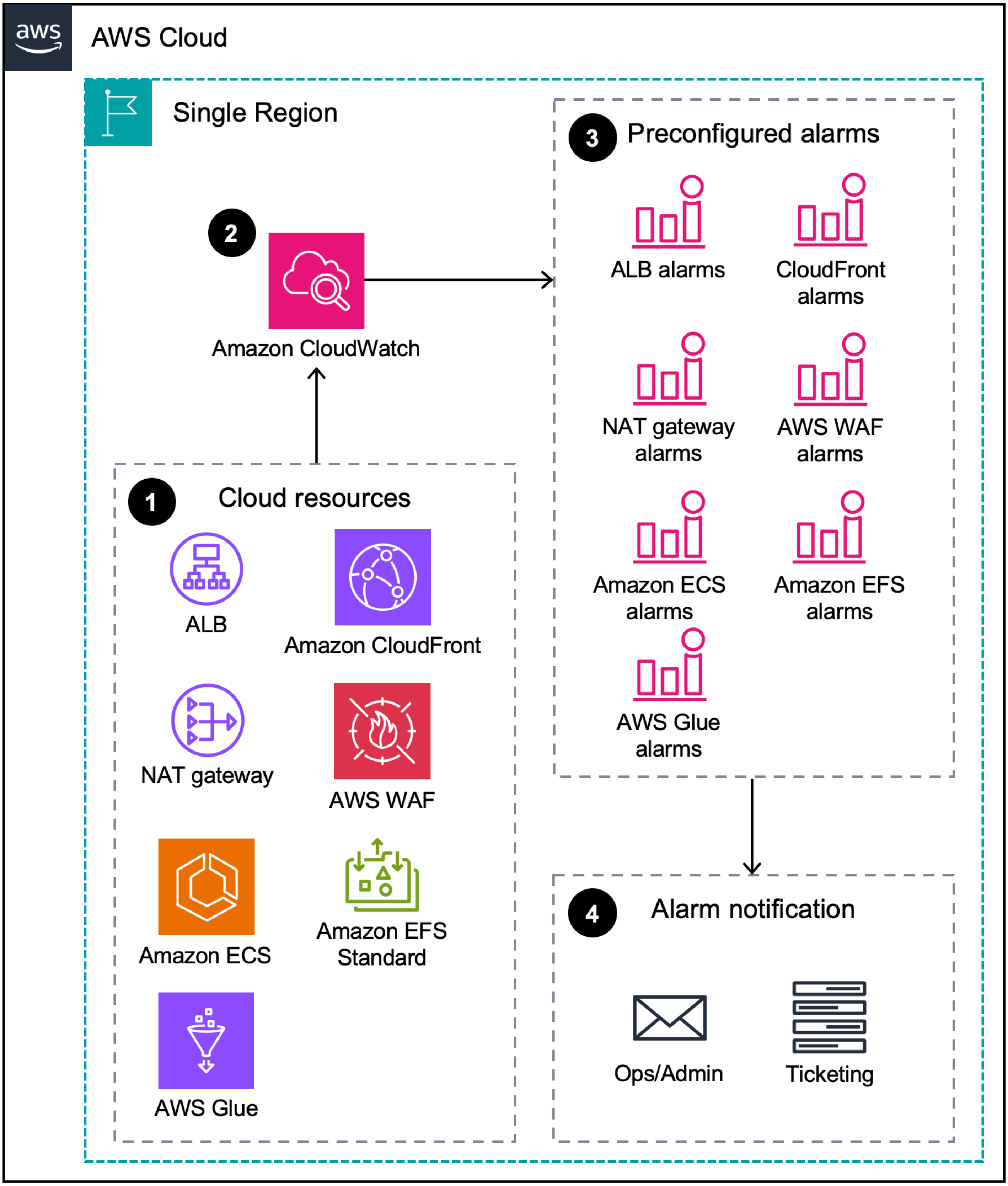
Amazon CloudWatch alarms monitor specific metrics in real time and proactively notify AWS Management Console users when predefined conditions are met. This solution has several CloudWatch alarms to help monitor its health and performance. In this section, each of the solution's alarms are listed with details on what metrics they track and what can invoke the alarms.

These alarms are enabled automatically when the AWS CDK stack is deployed. There are no further actions required to review the alarms.

> **Note**
>
> There are no subscriptions to alarm notifications by default. Add your team's email alias, paging address, or connection to an operational dashboard to be notified when an alarm changes state.

The following diagram shows the conceptual relationship between cloud resources created by this solution and pre-configured CloudWatch monitoring alarms.

Diagram showing overview of resources and their related CloudWatch alarms


AWS Cloud
Single Region
2
Amazon CloudWatch
3
Preconfigured alarms
ALB alarms
CloudFront alarms
NAT gateway alarms
AWS WAF alarms
Amazon ECS alarms
Amazon EFS alarms
AWS Glue alarms
1
Cloud resources
ALB
Amazon CloudFront
NAT gateway
AWS WAF
Amazon ECS
Amazon EFS Standard
AWS Glue
4
Alarm notification
Ops/Admin
Ticketing

Network traffic flow is monitored by ALB, CloudFront, NAT gateway, and AWS WAF alarms. ECS alarms focus on problems related to creating new instances. EFS alarms monitor throughput problems. Glue alarms change state on failures of the periodic AWS Glue job. The customer is responsible for subscribing to these alarms to a notification mechanism, such as email or text message.

AWS WAF

Blocked requests

- The alarm changes state if there is a large amount of blocked requests (greater than 75% of requests are blocked) within 1 minute.
- This alarm indicates that there is something wrong with the requests passing through the WAF or there could be malicious requests in the traffic.
- The alarm returns to the `OK` state if the data is within the acceptable threshold for 5 minutes.
- Metric: `BlockedRequests` > 75%

HTTP flood detected

- The alarm changes state if there is an HTTP flood attack detected within a 1-minute period.
- The alarm returns to the `OK` state if the data is within the acceptable threshold for 5 minutes.
- If detailed WAF logging is enabled, it will log the HTTP flood requests in the chosen destination. A datapoint will be logged in the CloudWatch metrics for the rule.
- Metric: `HttpFloodDetected` > 0

Allowed requests

- The alarm changes state if there is an anomaly in traffic with a high number of allowed requests within 1 minute.
- This alarm indicates a spike or burst in traffic.
- The alarm returns to the `OK` state if the data is within the acceptable threshold for 5 minutes.
- The alarm is an anomaly alarm and will form the threshold based on the previous history of the metric.

- Metric: `AllowedRequests` anomaly

CloudFront

Alarm: 5xx error rate

- The alarm changes state if there are any 500 type status codes. Reported as a percentage of total requests within a 1-minute period.
- This indicates a server failure. Check the CloudWatch logs to get further detail on the cause of the error.
- The alarm returns to the `OK` state if the error rate is within the acceptable threshold for 5 minutes.
- Metric: `5xxErrorRate` > 0%

Alarm: 4xx error rate

- The alarm changes state if greater than or equal to 1% of requests are 400 type status codes. Reported a percentage of total requests within a 1-minute period.
- This indicates a bad request or a possible configuration error. Check the CloudWatch logs to get further detail on the cause of the error.
- The alarm returns to the `OK` state if the error rate is within the acceptable threshold for 5 minutes.
- Metric: `4xxErrorRate` > 1%

Alarm: Requests

- The alarm changes state if there is an anomaly in traffic with a high number of requests within 1 minute.
- This indicates a spike or burst in traffic.
- The alarm returns to the `OK` state if the data is within the acceptable threshold for 5 minutes.
- The alarm is an anomaly alarm and will form the threshold based on the previous history of the metric.
- Metric: `Requests` anomaly

Application Load Balancer (ALB)

Target HTTP 4xx error rate

- The alarm changes state if there are `400` type status codes originating from the target (ECS). Reported as a percentage.
- This indicates a bad request or a possible configuration error. Check the CloudWatch logs to get further detail on the cause of the error.
- The alarm returns to the `OK` state if the error rate is within the acceptable threshold for 5 minutes.
- Metric: `HTTPCode_Target_4xxErrorRate` > 1%

Target HTTP 5xx error rate

- The alarm changes state if there are `500` type status codes originating from the target (ECS). Reported as a percentage.
- This indicates a server failure. Check the CloudWatch logs to get further detail on the cause of the error.
- The alarm returns to the `OK` state if the error rate is within the acceptable threshold for 5 minutes.
- Metric: `HTTPCode_Target_5xxErrorRate` > 0%

ALB HTTP 4xx error rate

- The alarm changes state if there are `400` type status codes originating from ALB. Reported as a percentage.
- This indicates a bad request or a possible configuration error. Check the CloudWatch logs to get further detail on the cause of the error.
- The alarm returns to the `OK` state if the error rate is within the acceptable threshold for 5 minutes.
- Metric: `HTTPCode_ELB_4xxErrorRate` > 1%

ALB HTTP 5xx error rate

- The alarm changes state if there are `500` type status codes originating from the target ALB. Reported as a percentage.
- This indicates a server failure. Check the CloudWatch logs to get further detail on the cause of the error.
- The alarm returns to the `OK` state if the data is within the acceptable threshold for 5 minutes.
- Metric: `HTTPCode_ELB_5xxErrorRate` > 0%

Target response time (Latency)

- The alarm changes state if there is a large amount of latency (greater than 100ms) reported within a 1-minute period.
- This could indicate a performance issue or scaling failure from ECS. Check the CloudWatch logs to get further detail on the cause of the error.
- The alarm returns to the `OK` state if the data is within the acceptable threshold for 5 minutes.
- Metric: `TargetResponseTime average` > 100 ms

Unhealthy host count

- The alarm changes state if there is a target that is considered unhealthy within a 1-minute period.
- The alarm returns to the `OK` state if the data is within the acceptable threshold for 5 minutes.
- Check the CloudWatch logs to get further detail on the cause of the error.
- Metric: `UnhealthyHotCount` > 0

NAT gateway

Port allocation errors

- The alarm changes state if there is a port allocation error in the NAT gateway.
- The alarm returns to the `OK` state if the data is within the acceptable threshold for 5 minutes.

- This can mean that too many concurrent connections are open through the NAT gateway and it caused a port allocation error.
- Metric: `ErrorPortAllocation` > 0

Packets dropped count

- The alarm changes state if a value greater than 0.01% is reached within a 1-minute period.
- This might indicate an ongoing transient issue with the NAT gateway.
- The alarm returns to the `OK` state if the data is within the acceptable threshold for 5 minutes.
- If this value exceeds 0.01 percent of the total traffic on the NAT gateway, check the AWS Service Health dashboard.
- Metric: `PacketsDropCount` > 0.01%

Elastic Container Service (ECS)

CPU and memory utilization

- The alarm changes state if the container CPU utilization or memory utilization exceed 70% within 1 minute.
- Our scaling policies' target is 50%. If these alarms change state, it means the solution's Auto Scaling is not working.
- You might need to check if Auto Scaling is turned on or adjust the Auto Scaling settings.
- The alarm returns to the `OK` state if the CPU utilization and memory utilization are within the acceptable threshold for 5 minutes.
- Metric: `CPUUtilization` > 70%, `MemoryUtilization` > 70%

Elastic File System (EFS)

Percent of I/O utilization

- The alarm changes state if the I/O utilization is consistently equal to or greater than 100% for 1 minute, indicating the need for additional capacity.

- The alarm returns to the OK state if the I/O utilization is within the acceptable threshold for 5 minutes.
- If this metric is at 100% often, then consider moving the application to an EFS using the Max I/O performance mode.
- Metric: `PercentIOLimit` > 100%

AWS Lake Formation permission errors

This solution is configured to use IAM permissions for all AWS AWS Glue Data Catalog resources. If you had previously configured your AWS account to use Lake Formation for all new database and tables prior to deploying the solution you might see the following error when the Metric ETL Glue Job attempts to run for the first time:

```
AccessDeniedException: An error occurred (AccessDeniedException) when calling the
 GetTable operation: Insufficient Lake Formation permission(s)
```

To fix this error without reverting your account-wide permissions back to the default settings, you must grant the Glue Job IAM role permission to access the solution database and table resources.

1. To grant the `MetricsEtlJobRole` **Super** permissions to all tables within the solution database, see Granting table permissions using the named resource method in the *AWS Lake Formation Developer Guide*.
2. Re-run any failed Glue jobs, making sure to pass in the `--object_keys` parameter with the failed parameter values from previous runs.

Get Further Assistance

If you have questions or need further assistance with the implementation of the solution guidance please reach out to your AWS Account team and chat with an AWS subject matter expert on this solution guidance.

Uninstall the solution

You can uninstall the Guidance for Deploying a Prebid Server on AWS solution from the AWS Management Console or by using the AWS Command Line Interface.

Using the AWS Management Console

1. Sign in to the AWS CloudFormation console.
2. On the **Stacks** page, select this solution's installation stack.
3. Choose **Delete**.

Using AWS Cloud Development Kit (AWS CDK)

Run blow command from the root of the project directory

```
$ cdk destroy
```

Deleting the Amazon S3 buckets

To prevent accidental data loss, this solution is configured to retain the solution-created Amazon S3 buckets when you delete its AWS CloudFormation stack. After uninstalling the solution, you can manually delete the S3 buckets if you do not need to retain their data. Each stack contains the following seven S3 buckets:

- `ALBAccessLogsBucket`
- `ArtifactsBucket`
- `CloudFrontAccessLogsBucket`
- `DockerConfigsBucket`
- `DataSyncMetricsBucket`
- `MetricsEtlBucket`
- `CloudTrailLoggingBucket`

Follow these steps to delete each Amazon S3 bucket.

1. Sign in to the Amazon S3 console.
2. Choose **Buckets** from the left navigation pane.
3. Locate the S3 buckets whose name begins with prefix, `<stack-name>` .
4. For each S3 bucket, select the bucket then choose **Empty** then **Delete**.

To delete the S3 bucket using AWS CLI, run the following command:

```
$ aws s3 rb s3://<bucket-name> --force
```

Use the solution

Querying metrics with Athena

Metrics collected from the Prebid Server application running on ECS are stored in the MetricsEtl S3 bucket for querying with Athena.

This section details information on the metric definitions, Glue table schemas, and example queries to get started.

Metric definitions

System metrics are captured using the Vert.x Metrics Service Provider Interface (SPI).

Auction metrics are captured both in general auction metrics as well as per-adapter metrics for bid-adapters that have been configured.

The full list of metrics with definitions can be viewed in the prebid-server-java GitHub repository.

Glue table schemas

Schemas for each table can be viewed in the Data Catalog in the AWS Glue console.

Example queries

The behavior of metric collection depends on the metric type. counter type metrics are flushed each time they are reported on and must be calculated across an entire time range for each container. For histogram, timer, and meter type metrics, the values are a snapshot of the container's lifetime at that particular timestamp. Athena views can be created to simplify pulling these tables in order to get the most up-to-date metrics across all containers. For more information about the various metrics types, see the Metrics project.

Create views

Create views for the various metrics types. Set up the following views for the histogram, timer, and meter tables. Replace the `<table-name>` variable for each table.

```
CREATE VIEW <table-name>_current_data AS
```

```
WITH timestamp_ranking AS (
SELECT
*,
RANK() OVER(PARTITION BY "container_id", "name" ORDER BY "timestamp" DESC) AS
 "timestamp_rank"
FROM "prebid-server-us-east-1-metricsetl-database"."<table-name>"
)
SELECT *
FROM timestamp_ranking
WHERE "timestamp_rank" = 1
```

Queries

Bid-Adapter metrics

1. Total number of bids received:

```
SELECT
"name",
SUM("count") AS "total_bids_received"
FROM "prebid-server-us-east-1-metricsetl-database"."counter"
WHERE "name" LIKE 'adapter.%.bids_received'
GROUP BY "name"
```

2. Average bid price:

```
WITH timestamp_ranking AS (
SELECT
*,
RANK() OVER(PARTITION BY "container_id", "name" ORDER BY "timestamp" DESC) AS
 "timestamp_rank"
FROM "prebid-server-us-east-1-metricsetl-database"."histogram"
), current_data AS (
SELECT
"name",
"count",
"mean"
FROM timestamp_ranking
WHERE
"name" LIKE 'adapter.%.prices'
AND "timestamp_rank" = 1
), total_count AS (
SELECT
"name",
SUM("count") AS "total"
FROM current_data
GROUP BY "name"
), weighted_values AS (
SELECT
current_data."name",
current_data."mean" * (CAST(current_data."count" AS double) /
          total_count."total" ) AS "weighted_value"
FROM current_data
LEFT JOIN total_count
ON current_data."name" = total_count."name"
)
SELECT
"name",
SUM("weighted_value") AS "average_bid_price"
FROM weighted_values
GROUP BY "name"
```

3. Win rate:

```
WITH total_impressions AS (
    SELECT SUM("count") AS "total_impressions"
    FROM "prebid-server-us-east-1-metricsetl-database"."counter"
    WHERE "name" IN ('imps_banner', 'imps_video', 'imps_audio', 'imps_native')
), adaptor_bids_received AS (
    SELECT
        "name",
        SUM("count") AS "total_bids_received"
    FROM "prebid-server-us-east-1-metricsetl-database"."counter"
    WHERE "name" LIKE 'adapter.%.bids_received'
    GROUP BY "name"
)
SELECT
    adaptor_bids_received."name",
    CAST(total_impressions."total_impressions" AS double) /
    CAST(adaptor_bids_received."total_bids_received" AS double) AS "win_rate"
FROM adaptor_bids_received
          CROSS JOIN total_impressions
```

4. Bid rate:

```
WITH request_per_adapter AS (
SELECT
SPLIT("name", '.')[2] AS "bid_adapter",
SUM("count") AS "count"
FROM "prebid-server-us-east-1-metricsetl-database"."counter"
WHERE
"name" LIKE 'adapter.%.requests.gotbids'
OR "name" LIKE 'adapter.%.requests.nobid'
OR "name" LIKE 'adapter.%.requests.badinput'
OR "name" LIKE 'adapter.%.requests.badserverresponse'
OR "name" LIKE 'adapter.%.requests.timeout'
OR "name" LIKE 'adapter.%.requests.unknown_error'
GROUP BY SPLIT("name", '.')[2]
), gotbid_per_request AS (
SELECT
SPLIT("name", '.')[2] AS "bid_adapter",
SUM("count") AS "gotbid_requests"
FROM "prebid-server-us-east-1-metricsetl-database"."counter"
WHERE
"name" LIKE 'adapter.%.requests.gotbids'
GROUP BY SPLIT("name", '.')[2]
)
SELECT
request_per_adapter."bid_adapter",
CAST(gotbid_per_request."gotbid_requests" AS double) /
 CAST(request_per_adapter."count" AS double) AS "bid_rate"
FROM request_per_adapter
LEFT JOIN gotbid_per_request
ON request_per_adapter."bid_adapter" = gotbid_per_request."bid_adapter"
```

5. Bid request responses:

```
WITH request_per_adapter AS (
SELECT
SPLIT("name", '.')[2] AS "bid_adapter",
SPLIT("name", '.')[4] AS "response",
SUM("count") AS "count"
FROM "prebid-server-us-east-1-metricsetl-database"."counter"
WHERE
"name" LIKE 'adapter.%.requests.gotbids'
OR "name" LIKE 'adapter.%.requests.nobid'
OR "name" LIKE 'adapter.%.requests.badinput'
OR "name" LIKE 'adapter.%.requests.badserverresponse'
OR "name" LIKE 'adapter.%.requests.timeout'
OR "name" LIKE 'adapter.%.requests.unknown_error'
GROUP BY SPLIT("name", '.')[2], SPLIT("name", '.')[4]
)
SELECT
request_per_adapter."bid_adapter",
"response",
"count"
FROM request_per_adapter
```

General auction metrics

1. Fill rate:

```
WITH total_impressions AS (
SELECT SUM("count") AS "total_impressions"
FROM "prebid-server-us-east-1-metricsetl-database"."counter"
WHERE "name" IN ('imps_banner', 'imps_video', 'imps_audio', 'imps_native')
), total_requests AS (
SELECT SUM("count") AS "total_requests"
FROM "prebid-server-us-east-1-metricsetl-database"."counter"
WHERE "name" = 'imps_requested'
)
SELECT CAST(total_impressions."total_impressions" AS double) /
 CAST(total_requests."total_requests" AS double) AS "fill_rate"
FROM total_requests
CROSS JOIN total_impressions
```

2. Average request time:

```
WITH timestamp_ranking AS (
SELECT
*,
RANK() OVER(PARTITION BY "container_id", "name" ORDER BY "timestamp" DESC) AS
 "timestamp_rank"
FROM "prebid-server-us-east-1-metricsetl-database"."timer"
), current_data AS (
SELECT
"name",
"count",
"mean"
FROM timestamp_ranking
WHERE
"name" LIKE 'request_time'
AND "timestamp_rank" = 1
), total_count AS (
SELECT SUM("count") AS "total"
FROM current_data
), weighted_values AS (
SELECT current_data."mean" * (
CAST(current_data."count" AS double)/ total_count."total") AS "weighted_value"
FROM current_data
CROSS JOIN total_count
)
SELECT SUM("weighted_value") AS "average_request_time"
FROM weighted_values
```

3. Sum of all impression types:

```
SELECT
"name",
SUM("count")
FROM "prebid-server-us-east-1-metricsetl-database"."counter"
WHERE "name" IN ('imps_banner', 'imps_video', 'imps_audio', 'imps_native')
GROUP BY "name"
```

Developer guide

This section provides the source code for the solution and additional details for testing.

Source code

Visit our GitHub repository to download the source files for this solution and to share your customizations with others.

The Guidance for Deploying a Prebid Server on AWS templates are generated using the AWS CDK. See the README.md file for additional information.

Metrics

This solution currently writes Prebid Server metrics log data to the mounted EFS, which is shared across containers, and ingests and archives the data using a variety of data management tools continuously or as a batch process. The patch file registers a new logger stream to periodically write metrics data from Prebid Server to files using the standard Java logging mechanism.

Testing

Functional tests

This solution includes a set of functional tests against the deployed solution to verify the basic operations of this solution. Users can run these tests by using the following steps:

1. Deploy this solution. If you have previously deployed this solution, skip this step.
2. Clone the solution’s repository.
3. Open a new terminal session and navigate to the `source/tests/functional_tests` directory.
4. Run the following command. Replace the `<stack-name>` and `<profile-name>` variables with name of the stack deployed in step 1 and profile name of AWS CLI credentials, respectively. Replace the `<in-venv-flag>` with 1 if the tests are running in an existing Python virtual environment or with 0 if not. See the README.md file of functional tests on the GitHub repository for further details.

```
$ sh run-functional-tests.sh [-h] [-v] [--test-file-name] [--extras] [--region] --
stack-name <stack-name> --profile <profile-name> --in-venv <in-venv-flag>
```

Distributed Load Testing (DLT)

The load testing tools based on JMeter and Distributed Load Testing on AWS are an optional part of the solution, and can be used to verify basic operations and the configuration installed can scale to the capacity required for successful operation.

Two JMeter test plans are included with the solution source code and can be used standalone with JMeter or used for large-scale, multi-Region load testing with Distributed Load Testing on AWS.

Prebid Server test plan

This test plan is placed in `source/loadtest/jmx/prebid_server_test_plan.jmx`. It uses several commercial bidding adapters in Prebid Server configured to respond in test mode. The bidding adapters do not make connections over the Internet when invoked this way and respond with fixed data. This test plan is suitable for verifying basic operations of the deployed stack are working.

To use this test plan, follow these steps:

1. Deploy this solution. If you have previously deployed this solution, skip this step.
2. Open the test plan `source/loadtest/jmx/prebid_server_test_plan.jmx` in JMeter.
3. Update the URL under **User Defined Variables** in JMeter console with the CloudFront endpoint of Prebid Server deployed in step 1.
4. Install the Distributed Load Testing on AWS solution.
5. Open the **Distributed Load Testing on AWS** console.
6. Upload the updated test plan in step 3 and start the tests.

Accessing Prebid Server logs from EFS

The following procedure describes how to access Prebid Server logs from EFS.

To create an EC2 instance that will function as a bastion host

1. Sign in to the [Amazon EC2 console](https://console.aws.amazon.com/ec2/).
2. Choose **Launch instance**.
3. Provide a name and leave the default Amazon Machine Image and instance type.
4. Choose **Key pair**, and select a key pair. Create one if you don't already have any.
5. Under **Network settings**, choose **Edit**.
6. Select `PrebidVpc`.
7. Select the public subnet that is the 10.8.0.0 network - `PrebidVpc/Prebid-PublicSubnet1`.
8. Select **Create security group** and use default settings.
9. Leave the default storage settings and choose **Launch instance**.

To enable incoming NFS connections to the EFS access point

1. Navigate to the Amazon EFS console.
2. In the navigation pane, choose **File systems**.
3. Open the EFS file system that is in the prebid stack.
4. Choose the **Network** tab and note the security group ID for the security group with `PrebidfsEfsSecurityGroup` in the name.
5. Navigate to the Amazon EC2 console.
6. In the navigation pane, choose **Security Groups**.
7. Open the solution's security group and choose **Edit inbound rules**.
8. Under **Edit inbound rules**:
   a. Choose **Add rule**.
   b. Under **Type**, select **NFS**.
   c. Under **Source**, select **Custom**.
   d. Enter `10.0.0.0/8` for **CIDR blocks**.
   e. Choose **Save rules**.
9. Return to the EFS system and choose **File system policy**.
10. Choose **Edit**, add `elasticfilesystem:ClientMount` to the list of allowed actions, and choose **Save**.

To mount the EFS file system

1. Navigate to the AWS CloudFormation console.
2. Open the solution's stack, select the **Resources** tab, and select the EFS file system.
3. Choose **Attach**.
4. Copy the NFS mount command.
5. SSH into the EC2 instance that you just created.
6. Make a mount point directory:

   ```
   sudo mkdir efs
   ```

7. Paste the NFS mount command that you copied earlier. It looks similar to this:

   ```
   sudo mount -t nfs4 -o
    nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport fs-
   xxxxxxxxxxxxxxxxx.efs.us-east-1.amazonaws.com:/ efs
   ```

Reference

This section includes information about an optional feature for collecting unique metrics for this solution, pointers to related resources, and a list of builders who contributed to this solution.

Anonymized data collection

This solution includes an option to send anonymized operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. When invoked, the following information is collected and sent to AWS:

- **Solution ID** - The AWS solution identifier
- **Unique ID (UUID)** - Randomly generated, unique identifier for each solution deployment
- **Timestamp** - Data-collection timestamp

We collect metrics directly from the various resources in the solution. These are filtered for the last 24 hours. The name of each metric comes from the service and is defined by that service. **DeleteEfsFiles** and **StartGlueJob** are our counts.

AWS owns the data gathered through this survey. Data collection is subject to the Privacy Notice. To opt out of this feature, complete the following steps before deploying using AWS CDK.

1. Locate the mappings.py in your local
2. set the value of the **send_anonymous_usage_data** input parameter in the ***init*** method to **False**
3. Follow the deployment steps documented in the readme.

Related resources

You can use the Distributed Load Testing on AWS solution to verify basic operations and the configuration installed can scale to the capacity required for successful operation.

Contributors

- Ian Downard
- Immanuel George

- Tom Gilman
- Cheryl Isler
- Andrew Marriott
- Alessandro Narciso
- Mike Olson
- Yang Qin
- Thyag Ramachandran
- Chip Reno
- Jim Thario
- Ranjith Krishnamoorthy

Revisions

Publication date: *November 2025*

Review the CHANGELOG.md in GitHub to track version-specific improvements and fixes.

Notices

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