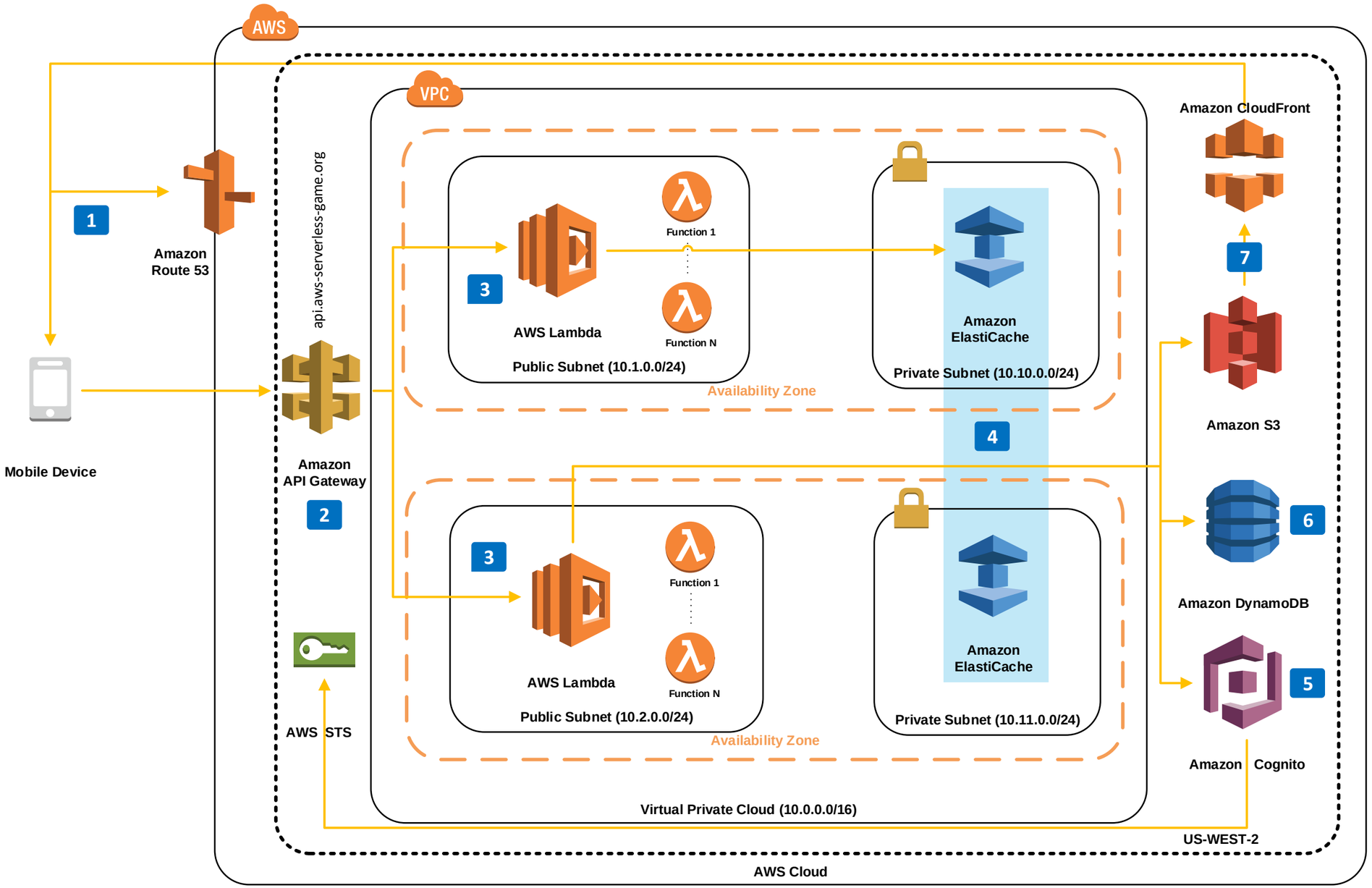
Serverless Core Game Backend

Power Your Games with No Servers

This serverless architecture powers the digital services for your online games. These workloads are a natural fit for running on Amazon Web Services, due to unexpected traffic patterns & highly demanding request rates. AWS provides the flexibility to start small & power up your architecture in response to your players. Let our managed serverless offerings handle the scaling and resource allocation. Use our managed services for popular caching & database technologies, & leverage this architecture that captures the best practices of some of the largest games running on AWS today.



1. The DNS requests are served by **Amazon Route 53**, a highly available Domain Name System (DNS) which routes your traffic to Amazon API Gateway.
2. API requests are routed to the **Amazon API Gateway** service which provides features such as security and throttling for your API's. Requests passing through the gateway can be transformed and passed through to backend server logic.
3. **AWS Lambda** provides the backend business logic for your game's digital services. The service is highly scalable and removes the need to run servers.
4. We run a Redis Cluster on **Amazon Elasticache** to power features like Leaderboards for our game. The managed aspect of Elasticache means you don't have to worry about failover and backups and automatic sharding makes it easy to scale your cluster.
5. Your users authenticate against **Amazon Cognito** integrated with your own custom backend authentication system. The credentials are then used for authorizing calls to the backend.
6. **Amazon DynamoDB** provides us with a fully managed, high speed No-SQL database. We leverage DynamoDB for a fast and consistent data persistence layer for our digital services.
7. Static content like DLC & Game Assets are delivered by our CDN **Amazon Cloudfront**, a global network of edge locations. The static content is stored in **Amazon Simple Storage Service (S3)**, a highly durable storage infrastructure for mission critical and primary data storage.

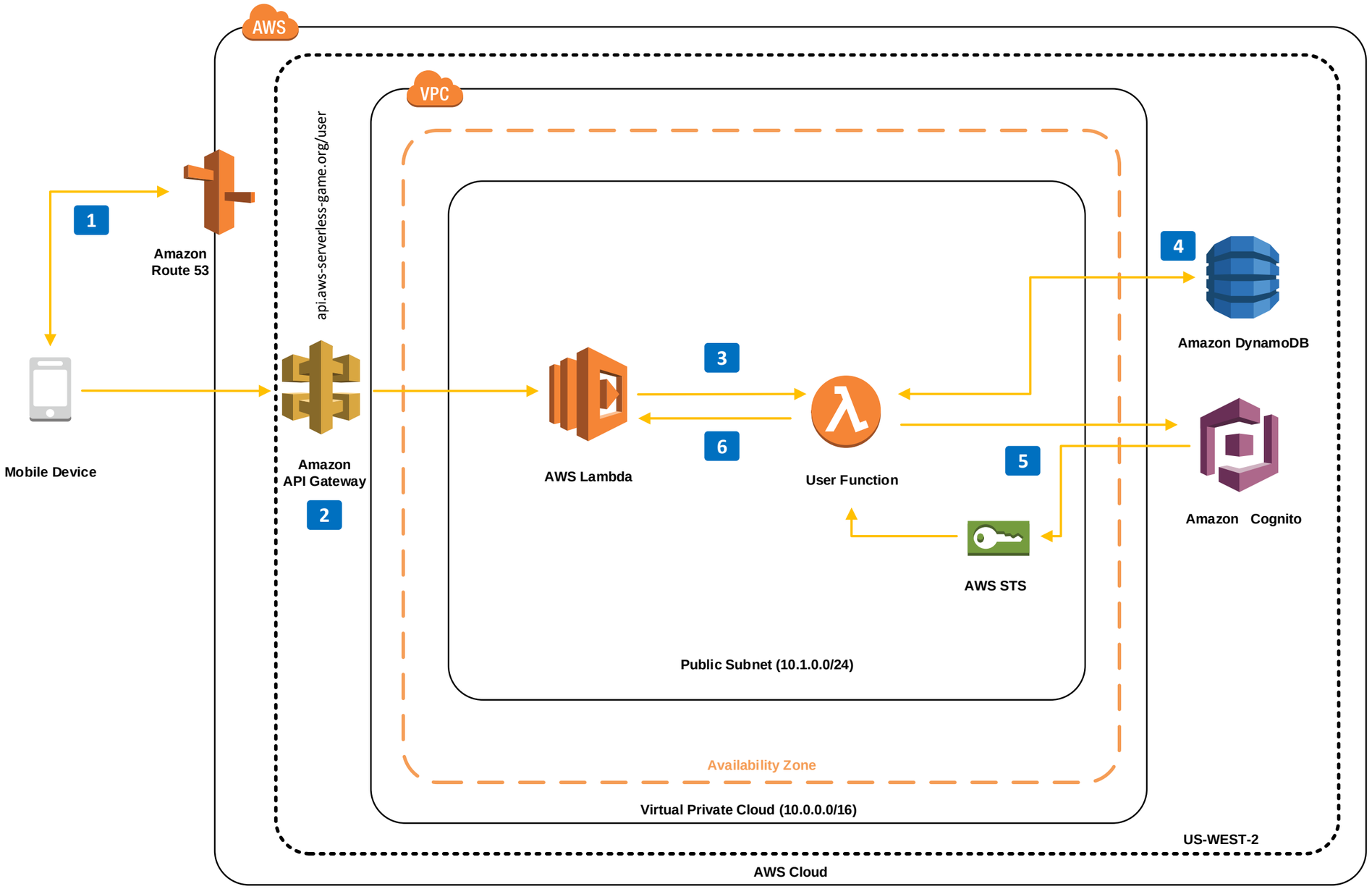
amazon web services

AWS Reference Architectures

Serverless Core Game Backend

Power Your Games with No Servers

This serverless architecture powers the digital services for your online games. These workloads are a natural fit for running on Amazon Web Services, due to unexpected traffic patterns & highly demanding request rates. AWS provides the flexibility to start small & power up your architecture in response to your players. Let our managed serverless offerings handle the scaling and resource allocation. Use our managed services for popular caching & database technologies, & leverage this architecture that captures the best practices of some of the largest games running on AWS today.



1. The DNS requests are served by **Amazon Route 53**, a highly available Domain Name System (DNS) which routes your traffic to Amazon API Gateway.
2. The game client makes a REST call to **/user** passing in a username and password value. The password value is hashed for security reasons. The data is sent in the POST body as a JSON object.
3. **API Gateway** passes the request to the **Amazon Lambda** function User triggering a custom event within the function that takes the JSON Body as a parameter. This function is configured with a timeout of 30 seconds and has 128MB of memory allocated to it.
4. The User Lambda function parses the values in the JSON body and then calls the **Amazon DynamoDB** service and checks for the existence of the user in the DynamoDB table **User.** If the user exists, the function then validates the password provided.
5. Once the user is authenticated, the User Lambda function leverages the **Developer Authenticated Identities** feature of **Amazon Cognito**. This feature lets developers user their own custom backend authentication service to get Amazon Cognito Credentials, that will then allow access to other AWS resources.
6. The User Lambda function calls the **GetOpenIdTokenForDeveloperIdentity** function to get an OpenId token. It appends it to the JSON body of the API response object which also contains the userid, username and other user metadata. The API call then returns a successful 200 response with the body.

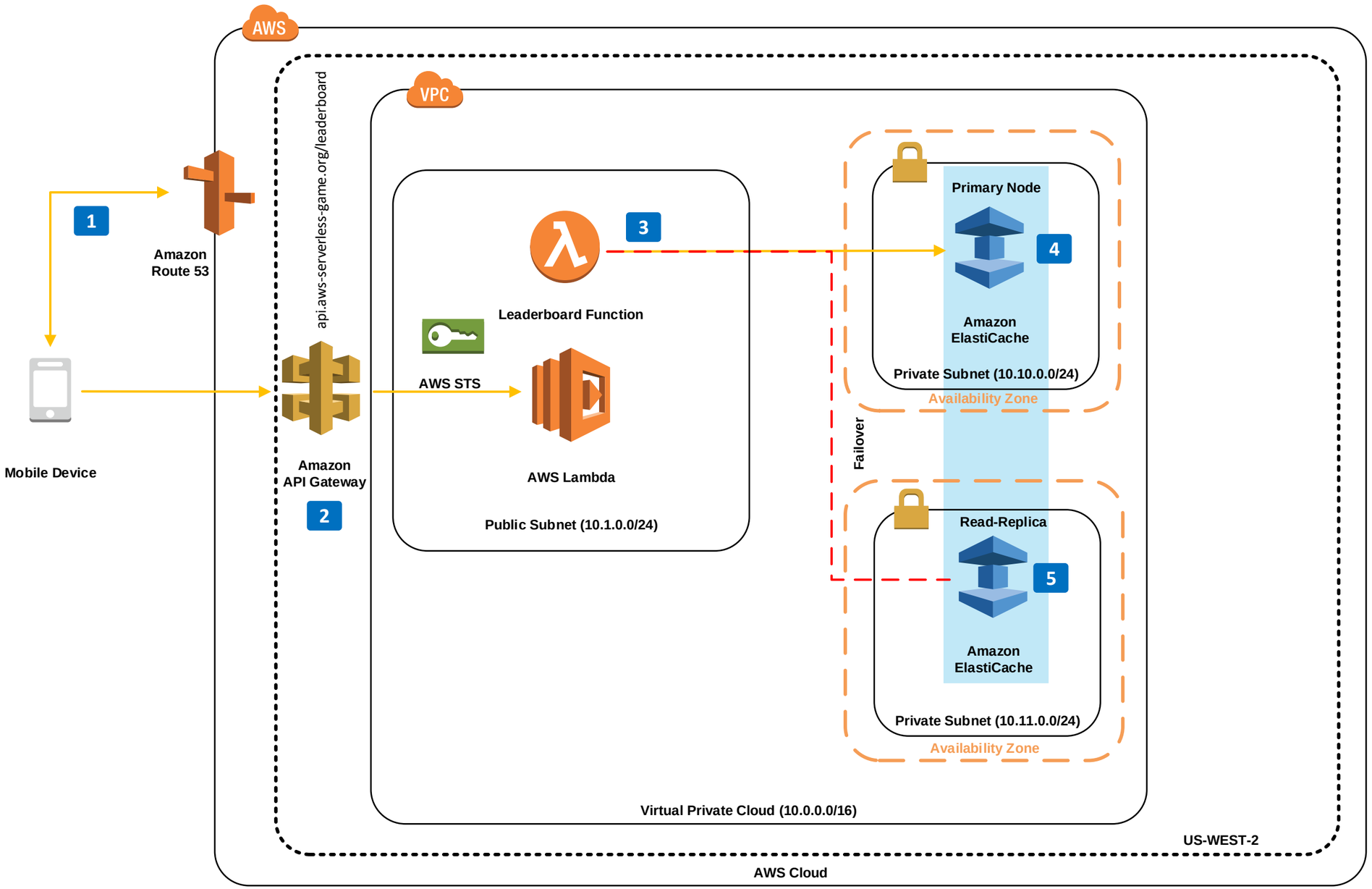
amazon web services

AWS Reference Architectures

Serverless Core Game Backend

Power Your Games with No Servers

This serverless architecture powers the digital services for your online games. These workloads are a natural fit for running on Amazon Web Services, due to unexpected traffic patterns & highly demanding request rates. AWS provides the flexibility to start small & power up your architecture in response to your players. Let our managed serverless offerings handle the scaling and resource allocation. Use our managed services for popular caching & database technologies, & leverage this architecture that captures the best practices of some of the largest games running on AWS today.



1. The DNS requests are served by **Amazon Route 53**, a highly available Domain Name System (DNS) which routes your traffic to Amazon API Gateway.

2. The game client makes a REST call to **/leaderboard** set of API calls. The API supports adding a new score in a POST call and retrieving the user's current rank in the leaderboard via a GET call. The client also has to pass in the Amazon Cognito Credentials it received when the user authenticated themselves, and append these credentials to the X-COGNITO-TOKEN header.

3. **API Gateway** passes the request to the **Amazon Lambda** function Leaderboard triggering a custom event within the function that takes the JSON Body as a parameter. This function is configured with a timeout of 30 seconds and has 128MB of memory allocated to it.

4. We run a Redis Cluster on **Amazon Elasticache** to power the Leaderboard feature for our game. The managed aspect of Elasticache means you don't have to worry about failover and backups and automatic sharding makes it easy to scale your cluster.

5. We have configured **Amazon Elasticache** to run in a **Multi-AZ** setup with a **Primary Node** in one AZ and a **Read Replica** in another AZ. In the event of a Primary failure, Elasticache promotes the Read Replica as the new primary and replaces the defunct Primary with a new node. The new node then becomes the new Read Replica.

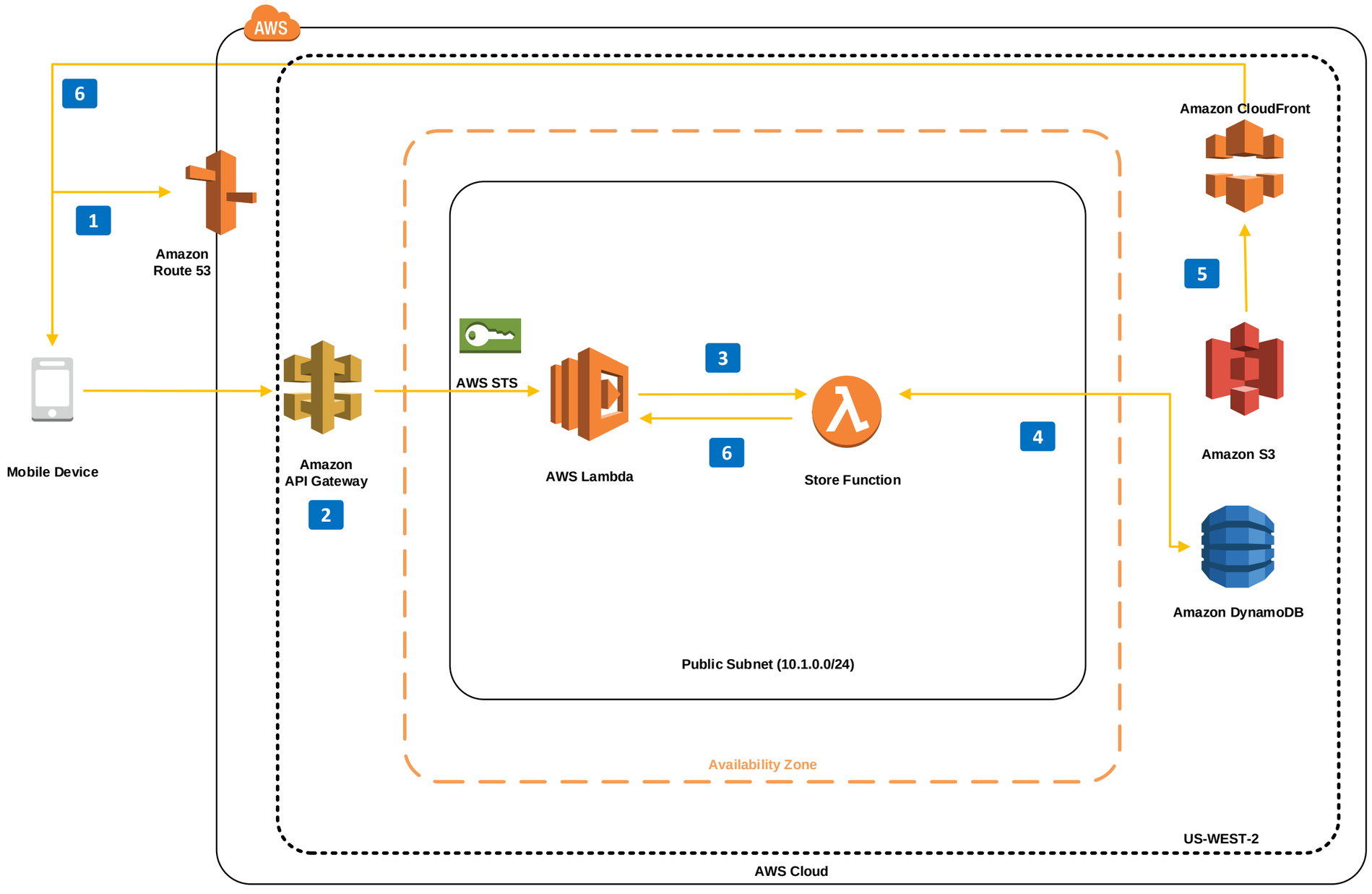
amazon web services

AWS Reference Architectures

Serverless Core Game Backend

Power Your Games with No Servers

This serverless architecture powers the digital services for your online games. These workloads are a natural fit for running on Amazon Web Services, due to unexpected traffic patterns & highly demanding request rates. AWS provides the flexibility to start small & power up your architecture in response to your players. Let our managed serverless offerings handle the scaling and resource allocation. Use our managed services for popular caching & database technologies, & leverage this architecture that captures the best practices of some of the largest games running on AWS today.



1. The DNS requests are served by **Amazon Route 53**, a highly available Domain Name System (DNS) which routes your traffic to Amazon API Gateway.
2. The game client makes a REST call to **/store** set of API calls. The API supports getting all stores items for a game, as well as details on individual store items. The client also has to pass in the Amazon Cognito Credentials it received when the user authenticated themselves, and append these credentials to the X-COGNITO-TOKEN header.
3. **API Gateway** passes the request to the **Amazon Lambda** function User triggering a custom event within the function that takes the JSON Body as a parameter. This function is configured with a timeout of 30 seconds and has 128MB of memory allocated to it.
4. The Store Lambda function parses the values in the JSON body and then calls the **Amazon DynamoDB** service and checks for items that exist for the game in the DynamoDB table **Store.** Once the items are retrieved, an array of Store Objects are returned.
5. One of the attributes of the Store object is an **Asset URL**. This URL link contains a link to the asset files associated with the store item (item images, DLC etc.). This url points to an object in an **Amazon S3** bucket which is the origin of all files.
6. **Amazon S3** is fronted by an **Amazon Cloudfront** distribution. Cloudfront's global network of edge locations that then allows the users to download the asset files from the closest location providing fast and reliable download speeds.

amazon web services

AWS Reference Architectures