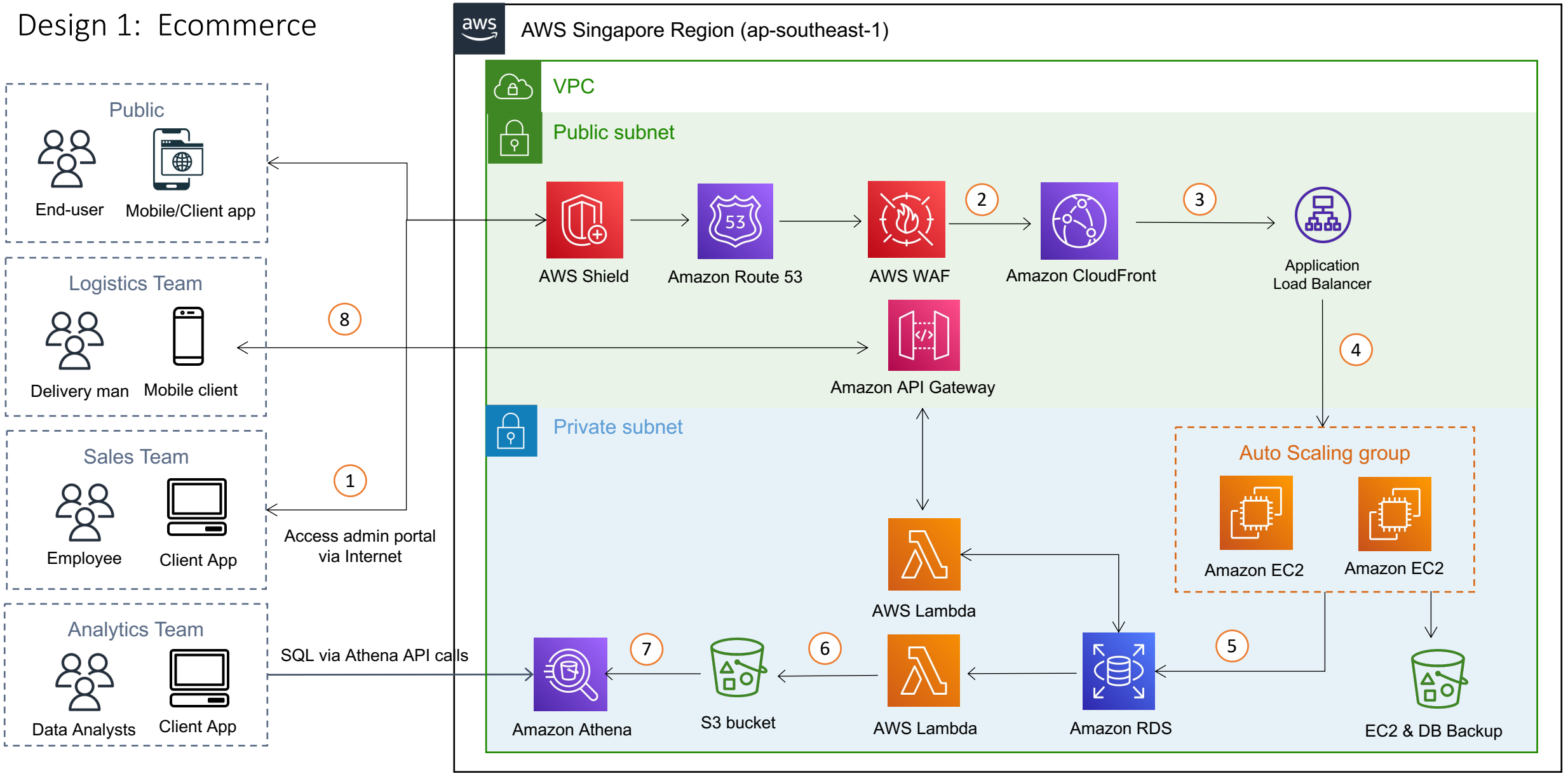
Section 3: System Design

Design 1: Ecommerce



Design 1: E-commerce

Assumptions made

- Assume that there is only one web application and one database created for this e-commerce setup
- Assume Sales Team has the appropriate access rights to access e-commerce admin portal
- Assume Analytics Team queries data via R/Python application hosted on-premise
- Assume AWS PrivateLink is established between on-premise and AWS Cloud environment
- Assume that GuardDuty, CloudWatch, CloudTrail services are all turned on for logging purposes (Depending on company's need)

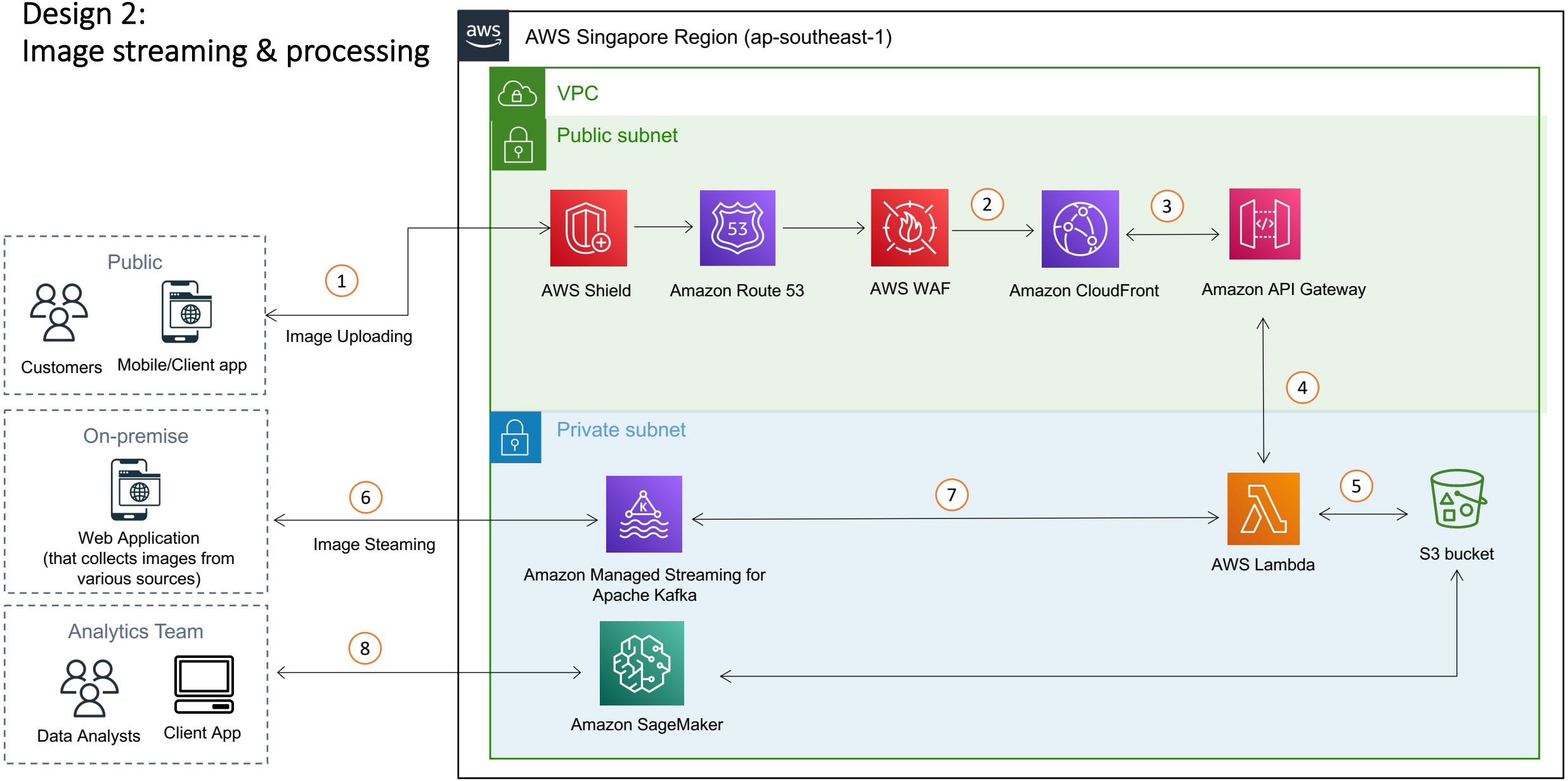
Note: The system architecture design is just a conceptual model. Development, Staging and Production environment is not well-defined here.

Design 1: E-commerce

General workflow

1. Sales Team accesses e-commerce admin portal via Internet to update items information in database. Amazon Route 53 first registers e-commerce domain and help to translate webpage URL to its respective IP addresses. AWS Shield is added to prevent Distributed Denial-of-service (DDOS) attack.
2. (optional) Amazon Cloudfront allows e-commerce company to serve variations in content to different country. It also provides caching features to improve application performance across a distributed geographic area. Web Application Firewall (WAF) is added to protect web application from common web exploits like cross-site scripting etc.
3. Application Load balancer serves as the single point of contact. The load balancer distributes incoming application traffic across multiple targets, such as EC2 instances. This increases the availability of the web application.
4. Auto-scaling is enabled to manage different workloads at different time of the year to cater to seasonal events such as year-end sales. EC2 instances will be spun up automatically when traffic is heavy and be removed when traffic is low.
5. As PostGreSQL (PSQL) database is selected as part of Q2 requirement, Amazon Relational Database Service (RDS) is chosen to store the web application data. This managed service is simple to setup, operate and scale in the cloud.
6. AWS Lambda function is created to query data from PSQL database on a regular basis. Data queried (e.g. Sales and membership status) will be stored in a S3 bucket.
7. Amazon Athena then picks up these information and avail it via Athena API. Data Analyst can perform their analysis by executing SQL statements through API calls, and results will be returned in their R/Python application. No write back to database is allowed in this case.
8. Logistic Team can access sales details via a REST API hosted on Amazon API gateway. Similar to Step(6), a lambda function will query order weight information and return the results to user when GET method is called. User can also update table for completed transactions via a POST method.

Design 2:
Image streaming & processing



Design 2: Image streaming & processing

Assumptions made

- Assume appropriate access rights are given to the Analytics team via IAM role.
- Assume AWS PrivateLink is established between on-premise and AWS Cloud environment
- Assume that GuardDuty, CloudWatch, CloudTrail services are all turned on for logging purposes (Depending on company's need)
- Assume data archival is performed by configuring S3 bucket Lifecycle policy
- Assume that Data Engineers will maintain all codes/scripts invoked on AWS Lambda
- Assume Amazon Cognito user pools is used to authenticate user identity
- Assume permissions are defined for each IAM role to access AWS resources

Note: The system architecture design is just a conceptual model. Development, Staging and Production environment is not well-defined here.

Design 2: Image streaming & processing

General workflow

1. Customers access image processing web application via Internet to upload images onto AWS cloud. Amazon Route 53 first registers web application domain and help to translate webpage URL to its respective IP addresses. AWS Shield is added to prevent Distributed Denial-of-service (DDOS) attack.
2. Amazon Cloudfront distribution provides a caching layer to reduce the cost of image processing and latency of subsequent image delivery. If the image is cached by CloudFront because of an earlier request, CloudFront will return the cached image instead of forwarding the request to the API Gateway. This reduces latency and eliminates the cost of reprocessing the image. Web Application Firewall (WAF) is added to protect web application from common web exploits like cross-site scripting etc.
3. Amazon API Gateway provides endpoint resources and initiates AWS Lambda function. Cross-Origin Resource Sharing (CORS) protection is implemented in API Gateway to restrict requests to only valid clients/sources, and authorize all requests based on configured authorizers.
4. Once requests are authorized, backend Lambda integration is invoked. Lambda function retrieves the original image from customer's existing S3 bucket and uses Sharp (an open-source image processing software) to return a modified version of the image to the API Gateway.
5. AWS Lambda stores the processed image on customer's individual S3 bucket. Note that Lambda execution environment runs only with a least-privileged IAM execution. The role grants the request access exclusively to the customer's S3 bucket.
6. Apache Kafka is used to ingest and store streaming data/images from web application, while serving reads for the web application powering the data pipeline.
7. AWS Lambda internally polls for new messages from the event source (i.e. Apache Kafka) and then synchronously invokes the target Lambda function. AWS Lambda reads the messages in batches and provides these to the Lambda function as an event payload. The maximum batch size is configurable. (The default is 100 messages.)
8. Analytics Team can train images and deploy training models using SageMaker. Sage Maker downloads / streams the training data from S3 buckets and run training algorithm on the data. SageMaker can also expose an endpoint for client application to send prediction requests.