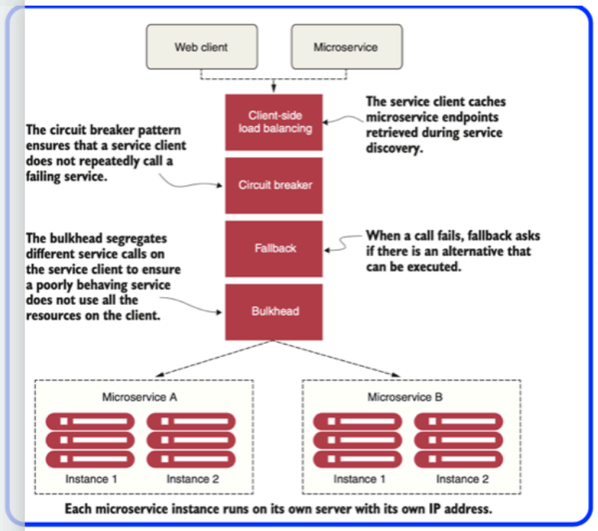
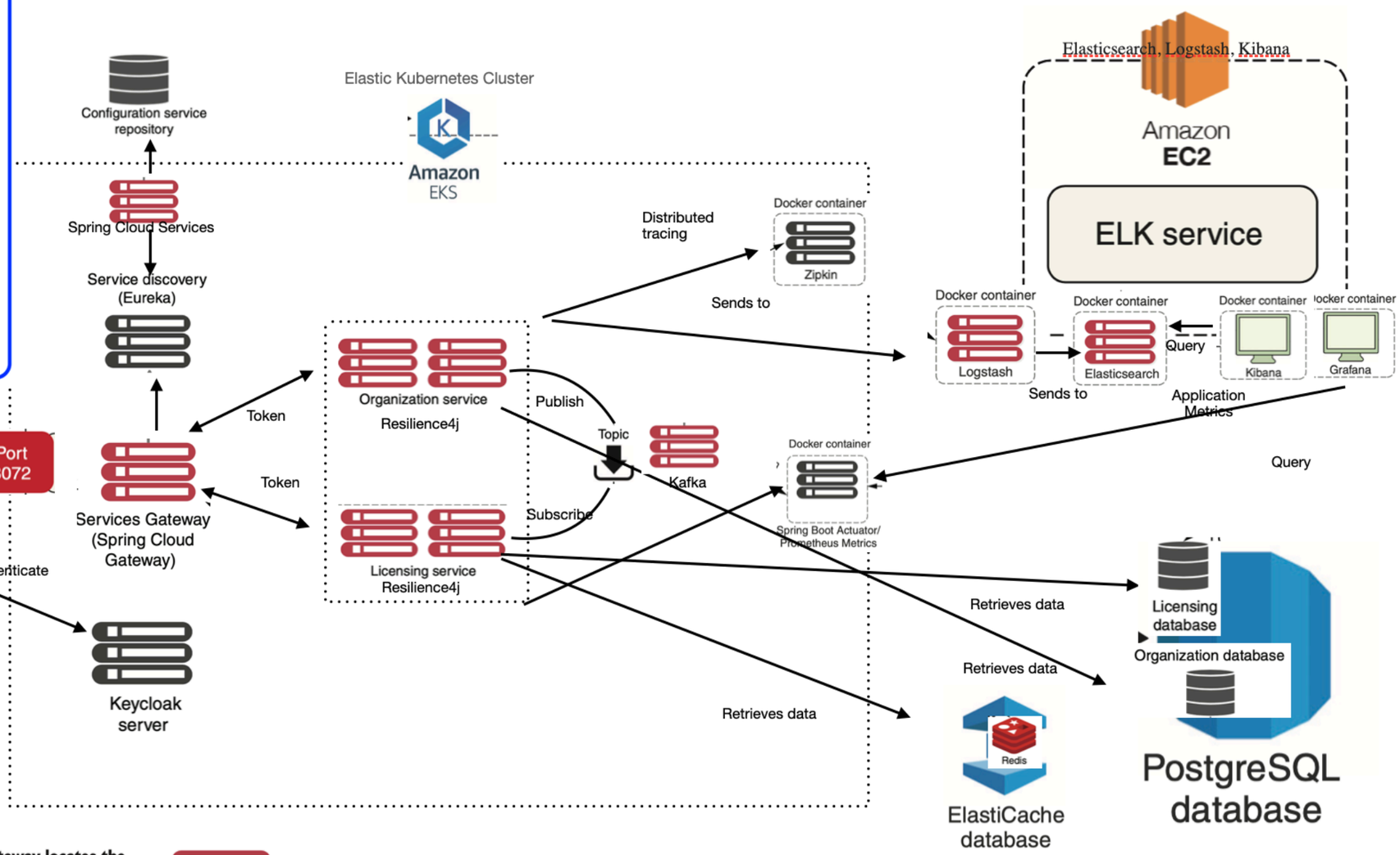
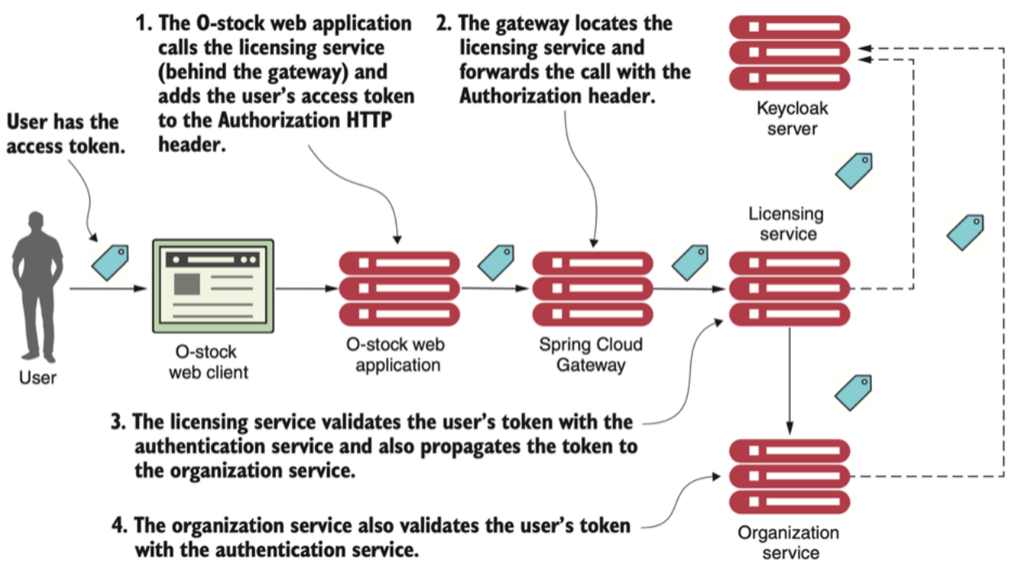
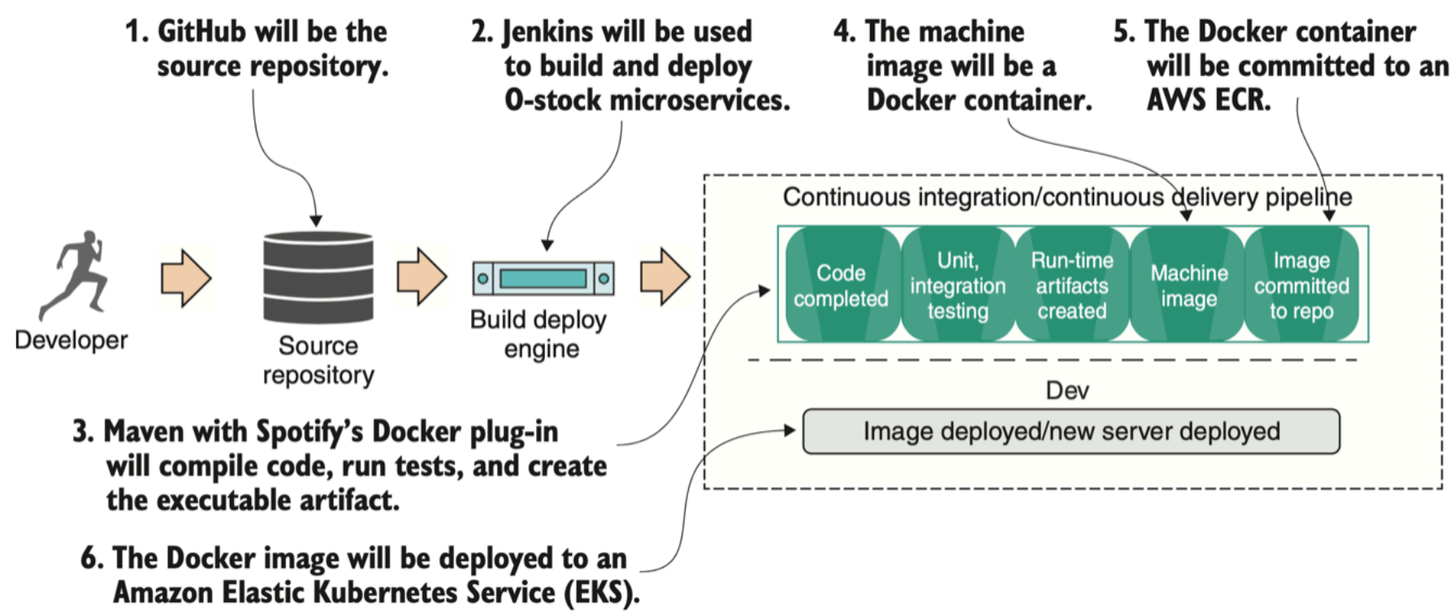

Web client
Microservice
Client-side load balancing
The service client caches microservice endpoints retrieved during service discovery.
The circuit breaker pattern ensures that a service client does not repeatedly call a failing service.
Circuit breaker
The bulkhead segregates different service calls on the service client to ensure a poorly behaving service does not use all the resources on the client.
Fallback
When a call fails, fallback asks if there is an alternative that can be executed.
Bulkhead
Microservice A
Microservice B
Instance 1
Instance 2
Instance 1
Instance 2
Each microservice instance runs on its own server with its own IP address.


Configuration service repository
Elastic Kubernetes Cluster
Amazon EKS
Spring Cloud Services
Service discovery (Eureka)
Elasticsearch, Logstash, Kibana
Amazon EC2
ELK service
Distributed tracing
Docker container
Zipkin
Sends to
Docker container
Logstash
Docker container
Elasticsearch
Query
Docker container
Kibana
Docker container
Grafana
Sends to
Application Metrics
Client
Token
Port 8072
Token
Token
Services Gateway (Spring Cloud Gateway)
Organization service
Resilience4j
Publish
Topic
Kafka
Subscribe
Licensing service
Resilience4j
Docker container
Spring Boot Actuator/ Prometheus Metrics
Query
Token
Authenticate
Keycloak server
Retrieves data
Licensing database
Organization database
Retrieves data
Retrieves data
Redis
ElastiCache database
PostgreSQL database


1. The O-stock web application calls the licensing service (behind the gateway) and adds the user's access token to the Authorization HTTP header.
2. The gateway locates the licensing service and forwards the call with the Authorization header.
User has the access token.
Keycloak server
Licensing service
User
O-stock web client
O-stock web application
Spring Cloud Gateway
3. The licensing service validates the user's token with the authentication service and also propagates the token to the organization service.
4. The organization service also validates the user's token with the authentication service.
Organization service


1. GitHub will be the source repository.
2. Jenkins will be used to build and deploy O-stock microservices.
4. The machine image will be a Docker container.
5. The Docker container will be committed to an AWS ECR.
Continuous integration/continuous delivery pipeline
Developer
Source repository
Build deploy engine
Code completed
Unit, integration testing
Run-time artifacts created
Machine image
Image committed to repo
Dev
Image deployed/new server deployed
3. Maven with Spotify's Docker plug-in will compile code, run tests, and create the executable artifact.
6. The Docker image will be deployed to an Amazon Elastic Kubernetes Service (EKS).