Blackbud : Case Study

- Created by Atlassian
- Developed Microservice architecture

① Architecture

Completely decentralized with many sub-microservices. Used multi-service architecture with Kubernetes platform

② Design Pattern

Used API Gateway pattern to provide a single entry point for all client requests and to route each request to the appropriate microservice.

Used Circuit Breaker pattern to prevent a network or service failure from cascading to other services.

③ Messaging Pattern

- Used publisher-subscriber pattern
- Used Spring Cloud Stream with RabbitMQ

Examples of independent microservices

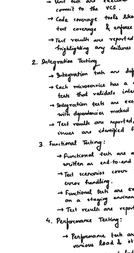
① Authentication Service
② Chat Room Service
③ Message Service
④ Notification Service

Testing Strategies in Microservices

* Types of Testing:-
  - 1) Unit Testing
    - → Jest
    - → Mocha
  - 2) Integration Testing
    - → Supertest
  - 3) Contract Testing
    - → Tests the communication path & interaction between microservices to ensure both services are correctly connected with no flaws.
    - → Pact
    - → Spring Cloud Contract
  - 4) End-to-End Testing
    - → Uses a complete flow from user perspective to cover all microservices involved in the flow.
    - → Cypress
    - → Selenium
    - → Apache JMeter
  - 5) Chaos Engineering
    - → Purposed by Netflix
    - → Intentionally introducing problems in your system to test its ability to withstand & recover.
    - → Netflix Chaos Monkey
    - → Chaos Mesh
  - 6) Functional Testing - Black box testing (black verification testing)

Microservice Testing Tools

- 1) Jest and Mocha
- 2) Rest-Assured
- 3) Spring Boot Test
- 4) Pact
- 5) Newman
- 6) Cypress
- 7) Chaos Monkey



Backup Pipeline for Microservice Testing

- → Incorporating automated testing practices at various stages of the pipeline to catch issues early, enable faster feedback loops, and facilitate flexible CI and CD.

- → Example :-
  1. Unit Testing:-
     - → Developer writes unit tests using JUnit for each microservice
     - → Unit tests are executed automatically on each code commit to the VCS
     - → Code coverage tools like JaCoCo are used to measure test coverage & enforce a minimum threshold.
     - → Test results are reported back to the dev team, highlighting any failures or regressions.
  2. Integration Testing
     - → Integration tests are defined using Spring Cloud Contract.
     - → Each microservice has a separate suite for integration tests that validate interactions with other microservices
     - → Integration tests are executed in an isolated environment, with dependencies mocked or stubbed
     - → Test results are reported, and any failures or compatibility issues are identified for resolution.
  3. Functional Testing:-
     - → Functional tests are automated using Selenium and written in end-to-end tests.
     - → Test scenarios cover main user flows, input validation, and error handling.
     - → Functional tests are executed against the deployed application on a staging environment.
     - → Test results are reported, highlighting any functional issues.
  4. Performance Testing:-
     - → Performance tests are created using JMeter and simulate various load & stress scenarios.
     - → Test scenarios include peak load, scalability, and endurance tests.
     - → Performance tests are executed regularly against the deployed microservices.
     - → Performance metrics such as response time, throughput, and resource utilization are collected & analyzed.
  5. Security Testing:-
     - → Security scans are performed using tools like SonarQube or OWASP ZAP.
     - → Vulnerability scans are executed regularly on the deployed microservices.
     - → Security issues, including known vulnerabilities or insecure configurations, are identified and reported.
     - → Security patches or remediation actions are taken based on the scan results.
  6. Deployment Testing
     - → Deployment tests are conducted as a part of the deployment pipeline.
     - → Deployments to different environments, such as staging and production, are automated using tools like Jenkins or GitLab CI/CD.
     - → Tests for environment configurations, deployment scripts, and deployment-specific operations are executed.
     - → Test results and deployment logs are monitored & reviewed closely to ensure successful deployments.

Overview of the deployment process

1. Setting up the VCS
   - → Setting up a Git or Bitbucket repo
   - → Create a dev branch & setting up branching strategy for feature branches, bug fixes, and releases.
2. Continuous Integration (CI)
   - → Use a CI tools like Jenkins or GitLab CI/CD to create a pipeline that triggers on code commits or pull requests.
   - → Configure the pipeline to execute the following stages:-
     - i) Build stage
       - ⊙ Use Maven or Gradle to build your Spring Boot application.
       - ⊙ Perform unit tests during the build process using a testing framework like JUnit.
       - ⊙ Generate code coverage reports using JaCoCo or a similar tool.
     - ii) Integration Testing Stage
       - ⊙ Set up a separate environment or use a tool like Docker to create isolated environments for integration testing.
       - ⊙ Configure the pipeline to deploy your microservices to this environment.
       - ⊙ Execute integration tests using tools like Spring Cloud Contract or REST-assured to validate interactions b/w microservices.
     - iii) Functional Testing Stage
       - ⊙ Use a tool like Selenium or Cypress to automate end-to-end functional tests.
       - ⊙ Write test cases that cover critical user flows, input validations, and error handling.
       - ⊙ Deploy the application on a staging environment and execute the functional tests against it.
     - iv) Performance Testing Stage
       - ⊙ Use a performance testing tool like Apache JMeter to create performance test scenarios.
       - ⊙ Configure the pipeline to deploy the app on a performance testing environment.
       - ⊙ Execute performance tests to measure response time, throughput, and scalability under different load scenarios.
     - v) Security Testing Stage
       - ⊙ Set up security scanning tools like OWASP ZAP or SonarQube to perform vulnerability scans.
       - ⊙ Configure the pipeline to trigger security scans against the deployed application.
       - ⊙ Review the scan reports & address any identified security vulnerabilities or issues.
     - vi) Deployment Stage
       - ⊙ Configure the pipeline to automatically deploy to different environments (staging, prod, etc.)
       - ⊙ Use containerization tools like Docker and orchestration frameworks like Kubernetes to simplify deployment.
       - ⊙ Execute deployment tests to ensure correct configuration, secret handling & environment setup.
3. Continuous Delivery (CD)
   - → Set up deployment configurations for different environments using configuration management tools like Ansible or Chef.
   - → Use configuration management tools like Ansible or Chef to automate the provisioning & configuration of the deployment environment.
   - → Integrate your CI/CD pipeline with a deployment platform like AWS CodeDeploy, Kubernetes, Google Cloud Platform, or Azure DevOps for smooth deployment.
4. Monitoring and Logging
   - → Set up monitoring & logging tools like Prometheus, Grafana, ELK Stack, or Splunk to monitor health & performance of your deployed microservices.
   - → Configure alerts and notifications to notify the development team of any issues or anomalies.