Project Context: PetCLinic Microservices -> AWS

Context and Scope

This project will migrate the Spring PetClinic Microservices demo from its local/on-premise setup to AWS Cloud. The focus is infrastructure modernization, CI/CD automation, observability, and resilience but not application feature development.

Stakeholders

Role	Responsibility
Project Sponsor	Funding, final approval
Project Manager	Scheduling, stakeholder coordination
Cloud Architect	Architecture, service selection
Dev Lead	App changes for cloud readiness
DevOps Engineer	CI/CD, IaC, deployments, monitoring
Security Team	IAM, encryption
End Users / Demo Audience	Acceptance and usability feedback

Expectations

- No app feature development unless necessary for cloud deployment.
- AWS is the only target cloud

Objectives

- Run full PetClinic microservices on AWS with CI/CD.
- Observability: logs, metrics, traces for 100% of services.
- Cost target: keep monthly infra cost under a defined limit .
- Security: secrets encrypted, least-privilege IAM, HTTPS for all endpoints.

Deadlines

Milestone	Date
Project approval	Oct 27, 2025
CI/CD & Automation	Nov 3, 2025
Infrastructure	Nov 10, 2025
Data	Nov 17, 2025
Observability	Nov 24, 2025

Milestone	Date
Prep: Presentation, Demo, and Pre-defense	Dec 3, 2025

In Scope

Included items	Objective
Application	Only necessary changes (if applicable) to facilitate cloud integration
Infrastructure	Design and deploy a reproducible, cloud-native architecture
CI/CD Automation	Implement automated build, test, and deployment pipelines
Containerization	Adapt existing microservices to use AWS.
Monitoring & Logging	Centralized logs, metrics, and traces
Security & IAM	Least-privilege IAM roles, encryption, and subnet segmentation.
Backup & Recovery	Redundancy, failover, backup, BCP/DRP
Documentation	Architecture diagrams, specifications, and operational runbooks.

Out of Scope

Excluded items	Reason
Application feature or UI changes	Functionality remains unchanged.
Multi-cloud or hybrid deployment	Focus solely on AWS environment.
Cost-optimization	Addressed in a later project if necessary

Requirements

Functional requirements

Stakeholder / Role	Requirement	Description
Developers	Continuous Integration	Each merge must trigger automated build, test, and image creation.
	Local to Cloud Parity	Development environment must mirror AWS setup.
DevOps Engineers	Automated Deployment	CI/CD pipeline must deploy microservices to Staging and Prod environments automatically.
	Test Automation	Integration tests must run automatically in CI/CD pipeline.

Stakeholder / Role	Requirement	Description
	Infrastructure as Code	All AWS resources defined through configuration files
	Monitoring & Alerts	Centralized logging, metrics, and tracing for all microservices. Automated alerting for service downtime or threshold breaches.
	Scalability	Services must be scalable
Security Team	Access Control	Roles per service with least-privilege permissions.
	Secrets Management	All secrets stored securely.
Product / Management	Availability & Demo Readiness	System must be reliable and presentable for client or internal demos.
End Users (Demo Audience)	Stable Access	Web UI and APIs must remain responsive under typical load.

Non-Functional Requirements

Category	Requirement	Standard
Development	Local to Cloud parity	Docker Compose or local ECS simulation
Maintainability	IaC	Code stored in version control
Performance	Pipeline execution time	< 10 minutes per merge
	Scaling	Services can be scaled horizontally
	API / UI response	< 200 ms under normal demo load
Reliability	Deployment success rate	≥ 99% successful deployments
	Alert response	Alerts trigger within < 5 minutes of failure detection
	Error tolerance	< 0.1% failed requests
Availability	System uptime	≥ 99.9% uptime
Observability	Logs, Metrics, Traces	Centralized in monitoring solution
Security	Least-privileged Roles	Roles restricted per service; no default full-access policies
	Secret encryption	Secrets stored in AWS
Cost	Budget target	Monthly AWS cost ≤ defined cap

System Components — Spring PetClinic Microservices

Component	Role / Function	Dependencies
`spring-petclinic-admin-server`	Provides admin UI and dashboards	Microservices, Config Server
`spring-petclinic-api-gateway`	Routes external requests to microservices	Customers, Vets, Visits, GenAI services
`spring-petclinic-config-server`	Centralized configuration	Git repo
`spring-petclinic-customers-service`	Manages customer data	RDBMS, Config Server
`spring-petclinic-vets-service`	Manages veterinary staff	RDBMS, Config Server
`spring-petclinic-visits-service`	Manages pet visit records	RDBMS, Customers Service
`spring-petclinic-genai-service`	Optional AI chat-bot	Microservices, RDBMS
`spring-petclinic-discovery-server`	Service registry / discovery	All microservices
RDBMS	Persistent storage	Customers, Vets, Visits

Architecture and Specifications

Project

- Kanban as agile methodology
- Breakdown of work and phases:
    - Infrastructure Setup
    - Service Orchestration
    - Configuration Management
    - CI/CD Automation
    - Security
    - Resilience
    - Observability

Assignments:

Role	Responsibilities
Cloud Architect	Design AWS target architecture, network, and IAM structure
DevOps Engineer	Build CI/CD pipelines, container orchestration, monitoring setup
Dev Lead	Containerize services, modify configs for cloud compatibility
Database Engineer	Migrate data from local RDBMS to AWS RDS, manage schema updates

Role	Responsibilities
Security Team	Set up access and roles for services
Everyone	Validate deployments, pipeline runs, rollback testing
Project Lead	Manage Asana Kanban board, ensure alignment and progress tracking

Source Code

- Architecture Type: Microservices deployed via containers, managed by ECS, behind an AWS Application Load Balancer (ALB).
- Review via pull request process:
    - All commits merged via PRs.
    - Peer review required before merging.
- Vaildation: Run tests during pipeline build

CI/CD

- Goal: Automate the entire software delivery process for all PetClinic microservices via Jenkins

Development Cycle Stages

Stage	Description
01. Code & Merge	Developer writes code and merges it with the staging branch
02. Build	Compile, resolve dependencies
03. Unit Test	Run service-level tests
04. Containerize	Build Docker image for service
05. Security Scan	Security validation via Trivy
06. Push to Registry	Push validated image
07. Deploy to Staging	Deploy for validation
08. Integration tests	Validate service communication
09. Deploy to Production	Promote validated build
11. Observability Check	Validate monitoring and alerts

Jobs and environments

- Each microservices has his own Jenkins pipeline per environment.

Environment	Purpose	Infrastructure
Development (Local)	Local testing, feature validation	Docker Compose
Staging (AWS)	Integration and pre-prod testing	ECS/EKS (staging cluster), RDS (test DB)

Environment	Purpose	Infrastructure
Production (AWS)	Live system	ECS/EKS (prod cluster), RDS (prod DB)

Storage

Type	Service	Use / Description	IOPS / Performance	Volume / Size	Backup Strategy
1. Database (RDBMS)	Amazon RDS (MySQL)	Structured data for each microservice schema	3,000–6,000 (gp3 default) or provisioned as needed	20 GB per schema	Automated daily snapshots (14-day retention)
2. Block Storage	Amazon EBS (gp3)	EC2-hosted Jenkins & ECS servers	3,000 baseline	/	Not necessary
3. Object Storage	Amazon S3	Logs, backups, images	Standard or Infrequent Access tiers	/	Cross-region replication or versioning enabled

Data

1. Location

- Eu-central-1 region
- Place database (RDS) and services in the same region and AZs.

2. Replication / Distribution

Data Type	Replication / Distribution Strategy
RDS (Postgres/MySQL)	Multi-AZ synchronous replication
S3 (images, artifacts)	Automatic cross-AZ durability

3. Links / Access

Access type	Route
Internal	Microservices access RDS via private VPC links.
	Images in S3 accessed via IAM roles or pre-signed URLs.
External	ALB routes external requests.
	HTTPS enforced for secure data transfer.

Network

Location

- Eu-central-1 region
- Deploy services across multiple AZs for high availability.
- All microservices, databases, and supporting infrastructure live inside a single VPC.
- Isolate the network from public internet by default

Network Segmentation & Filtering

- Public subnets: ALB, NAT gateway.
- Private subnets: ECS, RDS.
- Security groups: Service-specific firewall rules
- Tweak default ACLs if necessary

Addressing

- VPC: `10.0.0.0/16`
- Public subnet: `10.0.1.0/24`
- Private subnet: `10.0.2.0/24`
- Every service/database gets a internal IP in the private subnet
- Only load balancer or NAT gateway have public IP

Compute

Nodes

Environment	Nodes	Notes
Staging	3 ECS container instances (EC2)	Handles staging microservices, mirrors production setup
Production / Live	3 ECS container instances (EC2)	Fixed-size cluster, no autoscaling to reduce costs
Scalability	N/A for autoscaling	Fixed node count to reduce cost but still allow horizontal scaling via ECS task count or manual node addition.

Container Management

Container Registry:

- Amazon ECR for all microservice Docker images.
- Each microservice image tagged by Git commit SHA.

Deployment Strategy:

- ECS tasks run one or more containers per node.
- Service definitions ensure each microservice has the desired number of tasks.
- Jenkins updates ECS service definition after build.

ECS Orchestration

Cluster Setup:

- One ECS cluster per environment (staging and production).
- EC2 launch type for fixed nodes.

Service Definitions:

- Each microservice has an ECS service with a desired task count.
- Service linked to ALB .

Security

Area	Implementation / Notes
1. Authentication, Authorization, Auditing (AAA)	Spring Security
	IAM roles restrict AWS access per service
	Auditing: Not relevant since we don't handle sensitive data
	CloudWatch for app/service logs
2. Code Security	Static analysis via SonarQube
	No hardcoded credentials
	Secrets in AWS Secrets Manager
	Dependency scanning via Dependabot
3. Traffic Security	HTTPS enforced via ALB
	Internal TLS optional for microservices
	Security groups restrict inbound/outbound ports
	Private subnets for internal services and databases
4. Instance / Container Security	Use minimal and updated AMIs
	Regular patching, no direct SSH (bastion-only)
	Containers run as non-root users
	Vulnerability scanning before deploy
	Secrets passed via IAM roles or ECS environment vars

Observability

Aspect	Tools	Notes
Metrics	**Prometheus**	Collect CPU, memory, and ECS task metrics from node exporters

Aspect	Tools	Notes
		If microservices expose `prometheus-metrics`, integrate directly.
	Grafana	Dashboards for system and service health
Logs	**AWS CloudWatch Logs**	ECS task logs streamed to CloudWatch
		Structured JSON logging for easy filtering and search.
Traces	**AWS X-Ray**	Trace API calls across microservices.
Alerts	**CloudWatch Alarms**	CloudWatch for infrastructure-level alerts (CPU, memory, ECS health)
	Grafana Alerts	Grafana alert rules for application metrics from Prometheus.
		Alerts via email or Slack webhook.

Continuity & Recovery

Aspect	Approach / Tooling	Notes
Redundancy	Multi-AZ deployment	RDS and ECS nodes deployed across multiple Availability Zones for high availability.
		Load balancer automatically routes traffic to healthy tasks.
Failover	AWS-managed failover	RDS Multi-AZ provides automatic database failover.
		ECS services automatically restart failed tasks on healthy nodes.
		Manual intervention only needed for regional failures.
Backup	AWS Backup / RDS Snapshots	Automated RDS daily backups with retention policy.
	S3 Versioning	S3 bucket versioning for uploaded images and configs.
Business Continuity Plan	Operate from secondary region if needed	Documented procedure to restore environment in another AWS region using IaC templates (Terraform).
		Prioritize restoring RDS, Config Server, and API Gateway.
Disaster Recovery Plan	Cold standby in alternate region	No live duplication to save cost.
		Periodic replication of backups and images to secondary region.

Architecture Diagram

Solutions stack

Layer	Technologies / Services
Application Layer	Spring Boot microservices
Runtime / Platform Layer	Docker, Amazon ECS, Amazon ECR

Layer	Technologies / Services
CI/CD Layer	Jenkins, Gitea
Infrastructure Layer	Terraform, Ansible, Amazon EC2, VPC, subnets, security groups
Database / Storage Layer	Amazon RDS (MySQL), Amazon S3, Amazon EBS
Observability Layer	Prometheus, Grafana, CloudWatch
Security Layer	AWS IAM, Security Groups, HTTPS via ALB, Secrets Manager
Continuity & Recovery Layer	RDS automated snapshots, S3 versioning/replication, multi-AZ RDS, Terraform for redeploy
Network & Delivery Layer	Application Load Balancer (ALB), Route 53, NAT Gateway, Internet Gateway