**AWS Containerized Platform - Business Onboarding Guide**

**Table of Contents**

1. [Overview](https://claude.ai/chat/4e9f14f6-4fcc-4efa-b81a-2583759cd0de#overview)
2. [Prerequisites and Requirements](https://claude.ai/chat/4e9f14f6-4fcc-4efa-b81a-2583759cd0de#prerequisites-and-requirements)
3. [Phase 1: Account Setup and Access Requests](https://claude.ai/chat/4e9f14f6-4fcc-4efa-b81a-2583759cd0de#phase-1-account-setup-and-access-requests)
4. [Phase 2: Network and Security Configuration](https://claude.ai/chat/4e9f14f6-4fcc-4efa-b81a-2583759cd0de#phase-2-network-and-security-configuration)
5. [Phase 3: GitHub Repository Setup](https://claude.ai/chat/4e9f14f6-4fcc-4efa-b81a-2583759cd0de#phase-3-github-repository-setup)
6. [Phase 4: Application Onboarding](https://claude.ai/chat/4e9f14f6-4fcc-4efa-b81a-2583759cd0de#phase-4-application-onboarding)
7. [Phase 5: Deployment and Testing](https://claude.ai/chat/4e9f14f6-4fcc-4efa-b81a-2583759cd0de#phase-5-deployment-and-testing)
8. [Phase 6: Go-Live and Monitoring](https://claude.ai/chat/4e9f14f6-4fcc-4efa-b81a-2583759cd0de#phase-6-go-live-and-monitoring)
9. [Support and Troubleshooting](https://claude.ai/chat/4e9f14f6-4fcc-4efa-b81a-2583759cd0de#support-and-troubleshooting)
10. [Appendices](https://claude.ai/chat/4e9f14f6-4fcc-4efa-b81a-2583759cd0de#appendices)

**Overview**

This guide walks you through the complete process of onboarding your application to our AWS containerized platform. The platform provides reusable infrastructure components including load balancers, container orchestration, and CI/CD pipelines to accelerate your application deployment.

**What You'll Get**

* **Scalable Infrastructure**: Auto-scaling containerized applications
* **CI/CD Pipeline**: Automated build, test, and deployment workflows
* **Monitoring & Logging**: Comprehensive observability and alerting
* **Security**: Enterprise-grade security controls and compliance
* **Cost Optimization**: Efficient resource utilization and cost tracking

**Timeline Expectations**

* **Total Onboarding Time**: 2-4 weeks
* **Account Setup**: 3-5 business days
* **Network Configuration**: 5-7 business days
* **Repository Setup**: 1-2 business days
* **Application Deployment**: 3-5 business days
* **Testing & Go-Live**: 2-3 business days

**Prerequisites and Requirements**

**1. Business Requirements**

* [ ] **Application Assessment** completed
* [ ] **Architecture Review** approved by Cloud Center of Excellence
* [ ] **Budget Approval** for AWS resources
* [ ] **Compliance Requirements** documented (SOX, PCI, HIPAA, etc.)
* [ ] **Business Sponsor** identified and assigned

**2. Technical Requirements**

* [ ] **Application Containerization**: Application must be containerizable
* [ ] **12-Factor App Compliance**: Stateless application design preferred
* [ ] **Environment Variables**: Configuration through environment variables
* [ ] **Health Endpoints**: Application must expose health check endpoints
* [ ] **Logging**: Application logs to stdout/stderr

**3. Team Requirements**

* [ ] **Technical Lead** assigned
* [ ] **DevOps Engineer** (if available)
* [ ] **Product Owner** identified
* [ ] **Security Contact** designated

**4. Documentation Required**

* [ ] **Application Architecture Diagram**
* [ ] **Data Flow Diagram**
* [ ] **Security Requirements Document**
* [ ] **Disaster Recovery Requirements**
* [ ] **SLA/SLO Requirements**

**Phase 1: Account Setup and Access Requests**

**Step 1.1: ServiceNow Account Creation Request**

**ServiceNow Catalog Item: "AWS Account Provisioning"**

**Required Information:**

Business Details:

- Business Unit: [Your Business Unit]

- Cost Center: [Cost Center Code]

- Project Name: [Project/Application Name]

- Business Justification: [Detailed justification]

- Expected Go-Live Date: [YYYY-MM-DD]

Technical Details:

- Environment Type: [Development/Staging/Production]

- Expected Monthly Cost: [USD Amount]

- Data Classification: [Public/Internal/Confidential/Restricted]

- Compliance Requirements: [List applicable standards]

Contacts:

- Business Sponsor: [Name and Email]

- Technical Lead: [Name and Email]

- Security Contact: [Name and Email]

**ServiceNow Request Process:**

1. **Submit Request**: Create ServiceNow ticket using "AWS Account Provisioning" catalog item
2. **Business Approval**: Business sponsor approves the request
3. **Security Review**: Security team reviews compliance requirements
4. **Cloud Team Review**: Cloud Center of Excellence validates technical requirements
5. **Account Provisioning**: AWS account created and configured
6. **Access Notification**: Email notification with account details and next steps

**Expected Timeline**: 3-5 business days

**Step 1.2: IAM Access Request**

**ServiceNow Catalog Item: "AWS IAM Access Request"**

**Access Levels Available:**

* **Developer Access**: Read-only access to development resources
* **DevOps Access**: Full access to development, limited production access
* **Admin Access**: Full access to all environments (requires additional approval)

**Required Information:**

User Details:

- Employee ID: [Employee ID]

- Business Justification: [Why access is needed]

- Access Level Required: [Developer/DevOps/Admin]

- Environments: [dev/staging/production]

- Duration: [Permanent/Temporary with end date]

Manager Approval:

- Manager Name: [Direct Manager]

- Manager Email: [Email Address]

**Expected Timeline**: 2-3 business days

**Phase 2: Network and Security Configuration**

**Step 2.1: Firewall Request**

**ServiceNow Catalog Item: "Network Firewall Rule Request"**

**Standard Application Ports:**

Inbound Rules:

- Port 80 (HTTP): ALB health checks

- Port 443 (HTTPS): Application traffic

- Port 8080 (Optional): Application-specific

Outbound Rules:

- Port 80/443: Internet access for package downloads

- Port 443: AWS API calls

- Database Ports: As per application requirements

- SMTP Ports: Email services (if required)

**Request Template:**

Network Configuration:

Source Networks:

- Corporate Networks: [IP Ranges]

- Partner Networks: [IP Ranges if applicable]

- Internet: [0.0.0.0/0 for public applications]

Destination:

- AWS Account ID: [From Step 1]

- VPC CIDR: [Will be provided by Cloud Team]

- Subnets: [Public/Private subnet ranges]

Services:

- Application Type: [Web/API/Database]

- Ports Required: [List all ports]

- Protocol: [HTTP/HTTPS/TCP/UDP]

- Business Justification: [Why each port is needed]

**Expected Timeline**: 5-7 business days

**Step 2.2: DNS and Certificate Requests**

**ServiceNow Catalog Item: "DNS and SSL Certificate Request"**

**DNS Requirements:**

Domain Configuration:

- Primary Domain: [app-name.company.com]

- Environment Subdomains:

- Development: [app-name-dev.company.com]

- Staging: [app-name-staging.company.com]

- Production: [app-name.company.com]

Certificate Requirements:

- Certificate Type: [Wildcard/SAN/Single Domain]

- Validity Period: [1/2/3 years]

- Certificate Authority: [Internal CA/Public CA]

**Expected Timeline**: 3-5 business days

**Phase 3: GitHub Repository Setup**

**Step 3.1: Repository Creation Request**

**ServiceNow Catalog Item: "GitHub Repository Creation"**

**Repository Structure:**

your-application-platform/

├── frontend/

│ ├── src/

│ ├── Dockerfile

│ ├── .github/workflows/

│ └── terraform/

├── backend/

│ ├── src/

│ ├── Dockerfile

│ ├── .github/workflows/

│ └── terraform/

├── shared/

│ ├── terraform/

│ └── .github/workflows/

└── docs/

**Request Information:**

Repository Details:

- Repository Name: [your-application-platform]

- Organization: [Your GitHub Organization]

- Visibility: [Private/Internal]

- Template: [Use containerized-app-template]

Team Access:

- Admin Users: [List GitHub usernames]

- Write Users: [List GitHub usernames]

- Read Users: [List GitHub usernames]

Branch Protection:

- Protected Branches: [main, develop]

- Required Reviews: [2]

- Status Checks: [CI/CD pipeline]

**Expected Timeline**: 1-2 business days

**Step 3.2: Repository Verification Checklist**

Once your repositories are provisioned, verify the following:

**Frontend Repository Verification**

* [ ] **Repository Access**: Can access frontend repository with correct permissions
* [ ] **Template Files**: Dockerfile, GitHub Actions workflows present
* [ ] **Terraform Modules**: Reference to shared infrastructure modules
* [ ] **Environment Configuration**: YAML configuration files for each environment
* [ ] **Secret Configuration**: GitHub secrets configured for AWS access

**Backend Repository Verification**

* [ ] **Repository Access**: Can access backend repository with correct permissions
* [ ] **Template Files**: Dockerfile, GitHub Actions workflows present
* [ ] **Terraform Modules**: Reference to shared infrastructure modules
* [ ] **Environment Configuration**: YAML configuration files for each environment
* [ ] **Secret Configuration**: GitHub secrets configured for AWS access

**Shared Repository Verification**

* [ ] **Repository Access**: Can access shared infrastructure repository
* [ ] **ALB/NLB Modules**: Load balancer Terraform configurations
* [ ] **Network Configuration**: VPC, subnets, security groups
* [ ] **Monitoring Setup**: CloudWatch, alerting configurations
* [ ] **GitHub Actions**: Infrastructure deployment workflows

**Verification Commands**

# Clone repositories

git clone https://github.com/your-org/your-application-platform.git

cd your-application-platform

# Verify submodules

git submodule update --init --recursive

# Check Terraform modules

terraform init

terraform validate

# Verify GitHub Actions

gh workflow list

gh secret list

**Phase 4: Application Onboarding**

**Step 4.1: Application Configuration**

**Environment Configuration Files**

**Frontend Configuration** (frontend/config/dev.yaml):

environment: development

aws\_region: us-east-1

aws\_account\_id: "123456789012"

application:

name: "your-app-frontend"

version: "1.0.0"

port: 3000

health\_endpoint: "/health"

ecs:

cluster\_name: "your-app-cluster"

service\_name: "your-app-frontend"

task\_cpu: 256

task\_memory: 512

desired\_count: 2

min\_capacity: 1

max\_capacity: 10

container:

image\_name: "your-app-frontend"

dockerfile: "./Dockerfile"

build\_context: "."

load\_balancer:

target\_group\_name: "your-app-frontend-tg"

health\_check\_path: "/health"

health\_check\_interval: 30

monitoring:

log\_group: "/aws/ecs/your-app-frontend"

log\_retention\_days: 30

enable\_container\_insights: true

tags:

Environment: "development"

Project: "your-application"

Owner: "your-team"

CostCenter: "your-cost-center"

**Backend Configuration** (backend/config/dev.yaml):

environment: development

aws\_region: us-east-1

aws\_account\_id: "123456789012"

application:

name: "your-app-backend"

version: "1.0.0"

port: 8080

health\_endpoint: "/api/health"

ecs:

cluster\_name: "your-app-cluster"

service\_name: "your-app-backend"

task\_cpu: 512

task\_memory: 1024

desired\_count: 2

min\_capacity: 1

max\_capacity: 20

container:

image\_name: "your-app-backend"

dockerfile: "./Dockerfile"

build\_context: "."

database:

enabled: true

engine: "postgres"

instance\_class: "db.t3.micro"

allocated\_storage: 20

secrets:

- name: "database\_url"

description: "Database connection string"

- name: "api\_key"

description: "External API key"

monitoring:

log\_group: "/aws/ecs/your-app-backend"

log\_retention\_days: 30

enable\_container\_insights: true

**Step 4.2: Dockerfile Creation**

**Frontend Dockerfile Example**

# Frontend Dockerfile

FROM node:18-alpine AS builder

WORKDIR /app

COPY package\*.json ./

RUN npm ci --only=production

COPY . .

RUN npm run build

FROM nginx:alpine

COPY --from=builder /app/dist /usr/share/nginx/html

COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \

CMD curl -f http://localhost/health || exit 1

CMD ["nginx", "-g", "daemon off;"]

**Backend Dockerfile Example**

# Backend Dockerfile

FROM node:18-alpine AS builder

WORKDIR /app

COPY package\*.json ./

RUN npm ci --only=production

COPY . .

RUN npm run build

FROM node:18-alpine

WORKDIR /app

COPY --from=builder /app/node\_modules ./node\_modules

COPY --from=builder /app/dist ./dist

EXPOSE 8080

HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \

CMD curl -f http://localhost:8080/api/health || exit 1

USER node

CMD ["node", "dist/server.js"]

**Step 4.3: Terraform Configuration**

**Using Shared Modules (frontend/terraform/main.tf):**

terraform {

required\_version = ">= 1.0"

backend "s3" {

bucket = "your-terraform-state-bucket"

key = "frontend/terraform.tfstate"

region = "us-east-1"

}

}

# Reference to shared infrastructure modules

module "shared\_infrastructure" {

source = "git::https://github.com/your-org/shared-terraform-modules.git//modules/shared?ref=v1.0.0"

environment = var.environment

project\_name = var.project\_name

aws\_region = var.aws\_region

vpc\_cidr = var.vpc\_cidr

}

# Frontend ECS Service

module "frontend\_service" {

source = "git::https://github.com/your-org/shared-terraform-modules.git//modules/ecs-service?ref=v1.0.0"

service\_name = var.frontend\_config.application.name

cluster\_id = module.shared\_infrastructure.ecs\_cluster\_id

vpc\_id = module.shared\_infrastructure.vpc\_id

private\_subnets = module.shared\_infrastructure.private\_subnet\_ids

container\_definitions = [

{

name = var.frontend\_config.application.name

image = "${var.aws\_account\_id}.dkr.ecr.${var.aws\_region}.amazonaws.com/${var.frontend\_config.container.image\_name}:${var.frontend\_config.application.version}"

port = var.frontend\_config.application.port

environment = [

{

name = "NODE\_ENV"

value = var.environment

}

]

log\_configuration = {

logDriver = "awslogs"

options = {

"awslogs-group" = "/aws/ecs/${var.frontend\_config.application.name}"

"awslogs-region" = var.aws\_region

"awslogs-stream-prefix" = "ecs"

}

}

}

]

target\_group\_config = {

port = var.frontend\_config.application.port

protocol = "HTTP"

health\_check\_path = var.frontend\_config.application.health\_endpoint

}

alb\_listener\_arn = module.shared\_infrastructure.alb\_listener\_arn

tags = var.common\_tags

}

**Phase 5: Deployment and Testing**

**Step 5.1: Initial Deployment**

**GitHub Actions Workflow Verification**

**Check Workflow Status:**

# View workflow runs

gh run list --repo your-org/your-application-platform

# View specific workflow

gh run view [RUN\_ID] --repo your-org/your-application-platform

# View workflow logs

gh run view [RUN\_ID] --log --repo your-org/your-application-platform

**Deployment Process:**

1. **Code Commit**: Push code to feature branch
2. **CI Pipeline**: Automated testing and building
3. **Security Scan**: Container and dependency scanning
4. **Build Image**: Build and push to ECR
5. **Deploy to Dev**: Automatic deployment to development
6. **Integration Tests**: Automated testing in dev environment
7. **Manual Approval**: Required for staging/production

**Step 5.2: Environment Testing**

**Development Environment Testing**

# Test application endpoints

curl -f https://your-app-dev.company.com/health

curl -f https://your-app-dev.company.com/api/health

# Check ECS service status

aws ecs describe-services \

--cluster your-app-cluster \

--services your-app-frontend your-app-backend

# Verify load balancer health

aws elbv2 describe-target-health \

--target-group-arn [TARGET\_GROUP\_ARN]

**Testing Checklist**

* [ ] **Application Startup**: Application starts without errors
* [ ] **Health Endpoints**: Health checks return 200 OK
* [ ] **Load Balancer**: ALB routes traffic correctly
* [ ] **Auto Scaling**: Service scales up/down as expected
* [ ] **Logging**: Application logs appear in CloudWatch
* [ ] **Monitoring**: CloudWatch metrics are populated
* [ ] **Database Connectivity**: Database connections successful (if applicable)
* [ ] **External Dependencies**: API calls to external services work

**Step 5.3: Performance Testing**

**Load Testing Configuration**

Load Test Parameters:

- Concurrent Users: 100

- Test Duration: 10 minutes

- Ramp Up Time: 2 minutes

- Target Endpoints:

- Frontend: https://your-app-dev.company.com

- Backend API: https://your-app-dev.company.com/api

Expected Results:

- Response Time p95: < 500ms

- Response Time p99: < 1000ms

- Error Rate: < 0.1%

- CPU Utilization: < 70%

- Memory Utilization: < 80%

**Phase 6: Go-Live and Monitoring**

**Step 6.1: Production Deployment Checklist**

**Pre-Production Checklist**

* [ ] **Security Review**: Security team sign-off completed
* [ ] **Performance Testing**: Load testing results approved
* [ ] **Disaster Recovery**: Backup and recovery procedures tested
* [ ] **Monitoring Setup**: All alerts and dashboards configured
* [ ] **Documentation**: Runbooks and troubleshooting guides complete
* [ ] **Team Training**: Operations team trained on application support

**Production Deployment Process**

1. **Change Request**: Submit production change request in ServiceNow
2. **Approval Workflow**: Business and technical approvals obtained
3. **Maintenance Window**: Schedule deployment during approved window
4. **Blue/Green Deployment**: Deploy using zero-downtime strategy
5. **Smoke Testing**: Immediate post-deployment verification
6. **Monitoring**: 24-hour enhanced monitoring period

**Step 6.2: Monitoring and Alerting Setup**

**CloudWatch Dashboards**

Dashboard Components:

- Application Performance:

- Request Rate

- Response Time (p50, p95, p99)

- Error Rate

- Infrastructure Metrics:

- CPU Utilization

- Memory Utilization

- Network I/O

- ECS Service Health

- Business Metrics:

- User Sessions

- Transaction Volume

- Feature Usage

**Alert Configuration**

Critical Alerts (PagerDuty):

- Application Down: 0% healthy targets for 5 minutes

- High Error Rate: >5% error rate for 10 minutes

- Response Time: p95 > 2000ms for 15 minutes

Warning Alerts (Email):

- CPU High: >80% for 20 minutes

- Memory High: >85% for 15 minutes

- Disk Space: >90% used

Info Alerts (Slack):

- Deployment Started/Completed

- Auto Scaling Events

- Certificate Expiration (30 days)

**Step 6.3: Post Go-Live Activities**

**24-Hour Monitoring Period**

* [ ] **Application Performance**: Monitor key performance indicators
* [ ] **Error Tracking**: Review error logs and rates
* [ ] **User Feedback**: Collect and review user feedback
* [ ] **Infrastructure Stability**: Monitor resource utilization
* [ ] **Security Monitoring**: Review security events and alerts

**30-Day Review**

* [ ] **Performance Review**: Analyze 30-day performance trends
* [ ] **Cost Analysis**: Review actual vs. projected costs
* [ ] **Optimization Opportunities**: Identify areas for improvement
* [ ] **Lessons Learned**: Document insights and recommendations
* [ ] **Team Retrospective**: Conduct post-implementation review

**Support and Troubleshooting**

**Support Contacts**

**Cloud Platform Team**

* **Email**: cloud-platform-support@company.com
* **Slack**: #cloud-platform-support
* **ServiceNow**: IT Service Management → Cloud Services
* **Hours**: 24/7 for production issues, business hours for general support

**DevOps Support**

* **Email**: devops-support@company.com
* **Slack**: #devops-support
* **Hours**: Business hours (8 AM - 6 PM EST)

**Common Issues and Solutions**

**Deployment Issues**

**Issue**: GitHub Actions workflow fails

# Check workflow logs

gh run view --log

# Common solutions:

# 1. Check AWS credentials

# 2. Verify Terraform state lock

# 3. Check resource quotas

# 4. Validate configuration files

**Issue**: ECS service fails to start

# Check ECS service events

aws ecs describe-services --cluster [CLUSTER] --services [SERVICE]

# Check task logs

aws logs get-log-events --log-group-name [LOG\_GROUP]

# Common solutions:

# 1. Check container health endpoint

# 2. Verify environment variables

# 3. Check security group configuration

# 4. Validate task definition

**Performance Issues**

**Issue**: High response times

# Check CloudWatch metrics

aws cloudwatch get-metric-statistics \

--namespace AWS/ApplicationELB \

--metric-name TargetResponseTime

# Solutions:

# 1. Scale up ECS tasks

# 2. Optimize application code

# 3. Check database performance

# 4. Review ALB configuration

**Escalation Process**

**Severity Levels**

* **Severity 1**: Production down, business impact
  + Response: 15 minutes
  + Resolution: 4 hours
  + Escalation: Immediate to on-call engineer
* **Severity 2**: Degraded performance, limited impact
  + Response: 1 hour
  + Resolution: 24 hours
  + Escalation: If not resolved in 4 hours
* **Severity 3**: Non-urgent issues, questions
  + Response: 4 hours
  + Resolution: 5 business days
  + Escalation: If not resolved in 2 business days

**Appendices**

**Appendix A: ServiceNow Request Templates**

**A.1 AWS Account Provisioning Template**

Title: AWS Account Provisioning - [Project Name]

Business Details:

- Business Unit: [Fill in]

- Cost Center: [Fill in]

- Project Name: [Fill in]

- Business Justification: [Fill in]

- Expected Go-Live Date: [Fill in]

Technical Details:

- Environment Type: [Development/Staging/Production]

- Expected Monthly Cost: [Fill in]

- Data Classification: [Fill in]

- Compliance Requirements: [Fill in]

Contact Information:

- Business Sponsor: [Name and Email]

- Technical Lead: [Name and Email]

- Security Contact: [Name and Email]

Additional Requirements:

- [List any special requirements]

**Appendix B: Configuration Examples**

**B.1 Production Configuration**

# Production configuration template

environment: production

aws\_region: us-east-1

aws\_account\_id: "production-account-id"

application:

name: "your-app-frontend"

version: "1.0.0"

port: 3000

health\_endpoint: "/health"

ecs:

cluster\_name: "your-app-production-cluster"

service\_name: "your-app-frontend"

task\_cpu: 1024

task\_memory: 2048

desired\_count: 3

min\_capacity: 3

max\_capacity: 50

autoscaling:

scale\_up\_cooldown: 300

scale\_down\_cooldown: 300

target\_cpu\_utilization: 70

target\_memory\_utilization: 80

monitoring:

log\_group: "/aws/ecs/your-app-frontend"

log\_retention\_days: 90

enable\_container\_insights: true

enable\_detailed\_monitoring: true

backup:

retention\_days: 30

backup\_window: "03:00-04:00"

maintenance\_window: "sun:04:00-sun:05:00"

**Appendix C: Troubleshooting Guide**

**C.1 Common Error Messages**

Error: "Task failed to start"

Cause: Health check failure

Solution: Check application health endpoint and logs

Error: "Service discovery timeout"

Cause: Network connectivity issues

Solution: Verify security groups and VPC configuration

Error: "Image pull failed"

Cause: ECR permissions or image not found

Solution: Check ECR repository permissions and image tags

Error: "Terraform state lock"

Cause: Concurrent deployments

Solution: Wait for current deployment to complete or break lock

**Conclusion**

This onboarding guide provides a comprehensive roadmap for leveraging our AWS containerized platform. Following these steps ensures a smooth, secure, and compliant deployment of your applications.

For additional support or questions not covered in this guide, please contact the Cloud Platform Team through the channels listed in the Support section.

**Remember**: This is a living document that will be updated as processes and procedures evolve. Always refer to the latest version available in our documentation portal.