Cloud-Native Applications in AWS: Architecture, Failure Modes, and Incident Management Runbooks

1. What Are Cloud-Native Applications in AWS?

Cloud-native applications in AWS are systems architected specifically to leverage the core capabilities of the cloud. They are built using technologies that provide scalability, flexibility, and resilience by default. In AWS, cloud-native design typically means using services like AWS Lambda, ECS/EKS (for containers), DynamoDB, RDS, S3, and API Gateway, among others.

Key Principles:

- Microservices Architecture
- Containers and Orchestration
- Serverless Functions
- Managed Services and APIs
- Infrastructure as Code (IaC)
- Observability (Logging, Metrics, Tracing)

Incident Management Runbooks and Failure Mode Rationale

Runbook 1: Serverless Web App – Lambda Function Timeout or Crash

Failure Mode

Lambda Function Timeout or Crash

Symptoms

- 502/504 errors via API Gateway
- CloudWatch shows "Task timed out"
- No response to downstream services

Diagnostics

- CloudWatch Logs for Lambda execution time/errors
- API Gateway execution logs and latency metrics
- Review Dead Letter Queue (DLQ) if configured
- Check invocation metrics: `Duration`, `Error`, `Throttles`

Causes

- Infinite loop or blocking I/O in function code
- High-latency call to a third-party API
- Misconfigured timeout (default 3s or too short for logic)

Actions

- Optimize code for performance
- Extend Lambda timeout (up to 15 minutes)
- Add retries with exponential backoff
- Isolate long-running tasks into Step Functions or queues

Remediation

- Use async event sources (e.g., SQS instead of direct API Gateway)
- Break processing logic into microsteps via Step Functions
- Add observability: structured logging, X-Ray tracing
- Implement circuit breakers on external service calls

SRE / DevOps Notes

- Set CloudWatch Alarms on `Duration` and `Error` thresholds
- Monitor cold start latency and consider Provisioned Concurrency
- Validate retry behavior to avoid recursive failures
- Include Lambda memory/duration tuning in CI/CD pipelines

Runbook 2: ECS Microservices Platform – Service Crash / Container OOMKill

Failure Mode

Service Crash / Container Out-of-Memory (OOMKill)

Symptoms

- ECS Task marked “Stopped” unexpectedly
- Logs show “OOMKilled” or exit code 137
- Load balancer targets marked unhealthy

- 5xx errors returned from microservice

Diagnostics

- Inspect ECS task logs and event history
- Use CloudWatch metrics: `MemoryUtilization`, `CPUUtilization`
- Check task definition resource limits and exit codes
- Confirm Docker container logs for stack trace or memory issues

Causes

- Memory leak or inefficient memory usage in the app
- Improper memory allocation in ECS Task Definition
- Panic, crash, or segmentation fault in app process

Actions

- Tune memory and CPU resource allocations
- Use APM tools to profile memory usage (e.g., Datadog, X-Ray)
- Enable ECS task autoscaling based on CPU/Memory metrics
- Enable container restart policies and health checks

Remediation

- Adjust memory limits and apply changes via new Task Definition revision
- Move to AWS Fargate for automatic resource allocation
- Implement container-level memory monitoring and alerting
- Introduce rolling deployments to prevent service-wide impact

SRE / DevOps Notes

- Automate ECS rollbacks via CodeDeploy lifecycle hooks
- Use ECS Service Discovery with retries for resilience
- Include container crash patterns in chaos testing
- Maintain golden signals dashboards for ECS health

Runbook 3: Real-Time Analytics Pipeline – Kinesis Stream Lag

Failure Mode

Kinesis Stream Lag / Consumer Throughput Failure

Symptoms

- Dashboard and analytics data delayed
- `GetRecords.IteratorAgeMilliseconds` increases rapidly
- Lambda consumers fall behind or throttle
- Alerts for ingestion/processing lag triggered

Diagnostics

- Monitor `IteratorAgeMilliseconds` and `ReadProvisionedThroughputExceeded`
- Check CloudWatch metrics for consumer Lambda or KCL app
- Inspect CloudWatch Logs for throttling or retries
- Validate shard count and consumer concurrency

Causes

- Slow consumer processing
- Insufficient shards to handle throughput
- Throttling due to misconfigured IAM or network issues
- Lambda hitting concurrency limits

Actions

- Increase number of shards via resharding
- Tune consumer batch size and buffer processing logic
- Enable Enhanced Fan-Out (EFO) to reduce contention
- Scale consumers horizontally (e.g., increase Lambda concurrency)

Remediation

- Use buffering layers like Kinesis Firehose
- Add SQS as an intermediary buffer if consumer backpressure persists
- Break complex processing into parallel Lambda chains
- Monitor shard utilization with custom metrics

SRE / DevOps Notes

- Alert on iterator age > 1 minute and throughput errors
- Run performance load tests to validate stream design
- Use DLQs to track dropped records
- Implement retry/backoff policies in consumers

Rationale for Chosen Failure Modes

The selected failure modes reflect some of the most operationally impactful and commonly encountered incidents in cloud-native AWS architectures:

1. **Lambda Timeout/Crash** – A frequent problem in serverless stacks where developers underestimate the duration or complexity of logic. This directly affects user experience via APIs.
2. **Container Crash / OOMKill in ECS** – Memory exhaustion or improper sizing is a top reason for microservice instability in containerized environments.
3. **Kinesis Stream Lag** – A key performance issue in real-time data pipelines, often resulting in SLA violations for analytics or alerting applications.

These failure modes were chosen for their relevance across both development and production environments and for their applicability to SRE, DevOps, and Production Support practices involving observability, scaling, resilience, and automation.

Incident Management Runbooks (Cont.)

Runbook 4: EKS – Pod Scheduling Failure

Failure Mode

Pod Scheduling Failure

Symptoms

- Pods stuck in Pending state
- Events show '0/3 nodes are available: insufficient memory/cpu'
- Application unavailable or degraded

Diagnostics

- Run 'kubectl describe pod <pod-name>'
- Check node pool utilization: 'kubectl top nodes'
- Inspect taints, tolerations, affinity rules
- Review Cluster Autoscaler logs

Causes

- Resource limits exceeded on nodes
- Taints or selectors mismatched
- Cluster Autoscaler not scaling
- Pod priority/preemption misconfiguration

Actions

- Scale up node group manually
- Adjust pod resource requests/limits

- Reconfigure affinity/toleration
- Drain and cordon faulty nodes

Remediation

- Implement autoscaling policies
- Use LimitRange defaults
- Enable pod priority classes
- Pre-deployment scheduling tests

SRE/DevOps Notes

- Monitor PendingPods, NodeAllocatableResources
- Use IaC for node pool definitions
- Alert on scheduling delays

Runbook 5: EKS – Service LoadBalancer Not Provisioned

Failure Mode

Service LoadBalancer Not Provisioned

Symptoms

- 'kubectl get svc' shows EXTERNAL-IP as <pending>
- ELB/ALB not created
- Application inaccessible externally

Diagnostics

- Check events with 'kubectl describe svc'
- Review Ingress/Service annotations
- Check cloud-controller-manager logs
- Validate IAM and subnet config

Causes

- IAM lacks LoadBalancer permissions
- Subnet tags missing or incorrect
- Bad annotations
- Network ACL or endpoint issues

Actions

- Fix subnet tags
- Verify IAM policy for nodes
- Recreate service with correct annotations
- Use tested Ingress templates

Remediation

- Automate IAM/subnet setup
- Validate service manifests in CI
- Use AWS Load Balancer Controller

SRE/DevOps Notes

- Monitor provisioning time
- Alert on stuck services
- Enforce IaC validation rules

Runbook 6: RDS – Too Many Connections

Failure Mode

Too Many Connections

Symptoms

- Application logs: 'too many connections'
- CloudWatch: spike in DatabaseConnections
- API returns 500 or timeouts

Diagnostics

- Query active connections (e.g., pg_stat_activity)
- Review app connection pool settings
- Check CloudWatch metrics

Causes

- No pooling or connection leaks
- Traffic surge
- Long idle connections
- Slow queries

Actions

- Restart apps to release connections
- Kill idle sessions
- Increase max_connections
- Scale DB or add replicas

Remediation

- Use PgBouncer or RDS Proxy
- Tune pool size and timeouts
- Profile queries and close idle sessions

SRE/DevOps Notes

- CloudWatch alerts on thresholds
- Use circuit breakers in app
- Test failover and pooling in staging

Runbook 7: RDS – Storage Exhaustion

Failure Mode

Storage Exhaustion

Symptoms

- App logs: 'disk full', 'cannot insert into...'
- CloudWatch: FreeStorageSpace alarms
- Writes fail, app degraded

Diagnostics

- CloudWatch metrics: FreeStorageSpace
- Query table sizes
- Check snapshot/transaction log size

Causes

- Rapid data growth
- No vacuuming (PostgreSQL)
- Backup accumulation
- No autoscaling

Actions

- Enable autoscaling
- Truncate/archive data
- Reindex or vacuum
- Manually scale up storage

Remediation

- Data retention policies
- Alerts on 80%, 90% thresholds
- Use Enhanced Monitoring

SRE/DevOps Notes

- Forecast growth in ops reports
- Review binlog/snapshot strategy

Runbook 8: S3 – Access Denied / Permission Errors

Failure Mode

Access Denied / Permission Errors

Symptoms

- Logs: 403 Forbidden, AccessDenied
- AWS CLI/API errors
- Uploads/downloads fail

Diagnostics

- Check IAM and bucket policies
- Use Access Analyzer
- Review CloudTrail AccessDenied events

Causes

- Missing s3:GetObject or similar perms
- Explicit Deny in SCP or bucket policy
- Wrong region/signature
- ACLs override policy

Actions

- Update IAM roles/policies
- Fix bucket/object permissions
- Test with 'aws s3api get-object'
- Re-sign requests

Remediation

- Apply least-privilege IAM
- Template policies
- Log access failures

SRE/DevOps Notes

- Use AWS Config to detect misconfigs
- Prevent drift via IaC

Runbook 9: S3 – Event Notification Failure

Failure Mode

Event Notification Failure

Symptoms

- No downstream action after upload
- Missing thumbnails, logs
- No recent events

Diagnostics

- Check S3 event config
- Validate Lambda/SNS/SQS
- Review logs and DLQs

Causes

- Misconfigured filters
- Missing Invoke permissions
- Throttling/quotas
- EventBridge misrouting

Actions

- Fix event config
- Add Lambda invoke permission
- Replay from DLQ
- Test events end-to-end

Remediation

- Use EventBridge for routing
- Monitor event failures
- Validate subscriptions in CI

SRE/DevOps Notes

- Add event test coverage
- Alert on missed triggers