Xgrid.co | Temporal

CERTIFIED
Temporal
Cloud
Partner

# The Temporal Production Deployment Checklist:

## Everything You Must Verify Before Shipping Temporal Workflows

www.xgrid.co

# 1. Infrastructure Strategy: Build vs Buy

### The Business Question:

- [ ] Where should engineering effort be spent: operating workflow infrastructure or building competitive advantage?

### The Risk:

- [ ] Months lost to infrastructure setup.
- [ ] Quarters spent correcting capacity and operational mistakes.
- [ ] Platform complexity distracts teams from product delivery while competitors move faster.

### The Decision Framework:

**Choose Temporal Cloud when:**

- Time-to-market outweighs infrastructure control
- Engineering resources are constrained (fewer than 3-4 dedicated platform engineers available)
- Compliance requirements fit managed-service boundaries
- Engineering velocity is prioritized over customization

**Choose Self-Hosted when:**

- Regulatory mandates explicitly require infrastructure control (air-gapped environments, specific audit capabilities)
- Existing platform engineering team already manages distributed systems at scale
- Unique requirements demand deep customization unavailable in managed services
- Strategic commitment exists to infrastructure operations as core competency

### Success Criteria:

- [ ] Decision documented with clear rationale aligned to business priorities
- [ ] Total cost of ownership calculated includes engineering time, not just platform fees
- [ ] Operational capacity assessed honestly—resources available today, not aspirational hiring
- [ ] Timeline established from decision to production-ready infrastructure

### Check Red Flags

- [ ] **Red Flag :01**
  We'll start self-hosted and migrate to Cloud later if needed" (migration is expensive)
- [ ] **Red Flag :02**
  We can hire platform engineers as we go" (infrastructure needs sustained investment)
- [ ] **Red Flag :03**
  Infrastructure decision made by individual contributor without business context

## 2. Worker Capacity: The Invisible Bottleneck

### The Business Question:

- [ ] How do we ensure workflows execute fast enough to meet customer expectations?

### The Risk:

- [ ] Workflows slow down mysteriously.
- [ ] Engineering adds server capacity—nothing improves.
- [ ] Months spent debugging while tasks queue invisibly.
- [ ] Customer experience degrades.
- [ ] Root cause sits in plain sight but no one's watching the right metric.

**Critical Insight**

Schedule-to-start latency is the single metric that reveals capacity problems before customers notice. Monitor this, and scaling becomes predictable. Miss it, and troubleshooting becomes guesswork.

## The Strategic Questions

**Scaling Strategy:**

- [ ] Can the system automatically add capacity when traffic spikes?
- [ ] Will Black Friday or product launches require manual intervention?
- [ ] Does the architecture support 10× growth without re-architecture?

**Workload Optimization:**

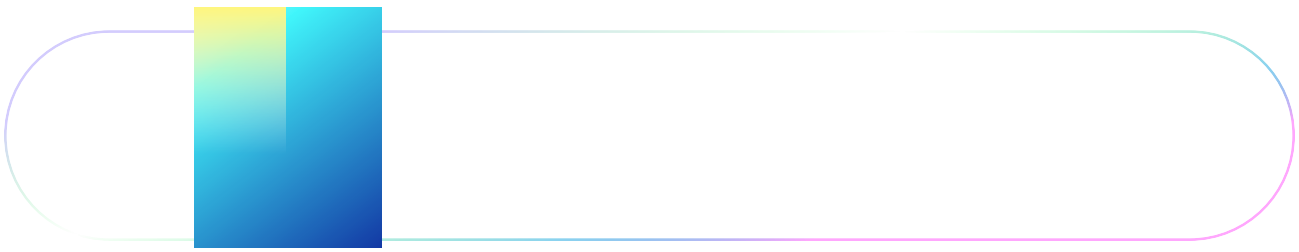
- [ ] Are CPU-intensive tasks separated from I/O-bound operations?
- [ ] Do high-priority customer workflows get reserved capacity?
- [ ] Can the team deploy to Kubernetes or equivalent orchestration platform?

## Success Criteria:

- [ ] Autoscaling configured based on actual capacity indicators, not generic CPU metrics
- [ ] Alert thresholds established that trigger before customer-facing impact
- [ ] Load testing completed simulating realistic traffic patterns and spikes
- [ ] Capacity headroom verified to handle unexpected growth

## Check Red Flags

- [ ] **Red Flag :01**
  Team monitoring only server metrics, not worker capacity
- [ ] **Red Flag :02**
  No automated scaling configured—manual intervention required for traffic spikes
- [ ] **Red Flag :03**
  All workflows sharing single worker pool regardless of priority or resource needs



# 3. Security Architecture: The Compliance Audit You Can't Fail

### The Business Question:

- [ ] Are we handling sensitive data in a way that satisfies auditors, regulators, and customers?

### The Risk:

- [ ] Compliance audit discovers customer data stored in plaintext.
- [ ] Engineering adds server capacity—nothing improves.
- [ ] Months spent debugging while tasks queue invisibly.
- [ ] Customer experience degrades.
- [ ] Root cause sits in plain sight but no one's watching the right metric.

### The Non-Negotiable Requirements

**Data Protection:**

- [ ] Workflow payloads must be encrypted before reaching Temporal infrastructure
- [ ] Encryption keys must remain under exclusive organizational control
- [ ] Engineers must debug workflows without exposing sensitive data broadly

**Workload Optimization:**

- [ ] All communication between workers and servers must be authenticated and encrypted
- [ ] Certificate management must be automated to prevent outages from expired credentials

### Success Criteria:

- [ ] End-to-end encryption validated—Temporal infrastructure never sees plaintext
- [ ] Secure debugging enabled—engineers can troubleshoot without security compromises
- [ ] Certificate automation configured—no manual renewal processes that create risk
- [ ] Compliance review completed—security team sign-off documented

### Check Red Flags

**Red Flag :01**

We'll add encryption after initial launch" (compliance violations start immediately)

**Red Flag :02**

Relying only on transport security (HTTPS) without payload encryption

**Red Flag :03**

Manual certificate management processes (operational overhead and outage risk)

## 4. Observability: Debugging Without Guesswork

### The Business Question:

When workflows fail at 3 AM, can we diagnose and fix the problem in minutes instead of hours?

### The Risk:

- Critical workflow fails & $500,000 transaction is lost.
- Engineering spends hours reconstructing what happened from scattered logs.
- Customer impact extends while the team guesses at the root cause.
- Meanwhile, competitors with proper observability fixed the same issue in 10 minutes.

### The Strategic Capabilities Required

**Incident Resolution:**

- Can engineers replay production failures locally under a debugger?
- Is there a complete audit trail of every workflow decision?
- Are metrics configured to detect degradation before customers notice?

**Proactive Monitoring:**

- Will the team know about capacity problems before they cause incidents?
- Are failure rate spikes detected automatically?
- Can operational queries answer "show me all stuck workflows for customer X"?

### Success Criteria:

- [ ] Event history replay validated—team trained on downloading and debugging production failures
- [ ] Metrics pipeline operational—Prometheus, Grafana, alerts configured
- [ ] Alert thresholds calibrated—early warning before customer impact
- [ ] Search capabilities enabled—operational queries work for business-relevant criteria

### Check Red Flags

- [ ] **Red Flag :01**
  Treating observability as "we'll add dashboards later"
- [ ] **Red Flag :02**
  No training on event history debugging—capability exists but team can't use it
- [ ] **Red Flag :03**
  Alerts configured only for outages, not degradation

## 5. Reliability Patterns: The $3 Million Double-Charge

### The Business Question:

- [ ] When failures happen—and they will—does the system recover safely or create cascading problems?

### The Risk:

- [ ] Network hiccup causes payment activity to fail. System retries.
- [ ] Customer charged twice. Then three times.
- [ ] 47 customers double-charged before discovery.

Refunds: $40,000. Customer service: $60,000. Reputation damage: incalculable.

### The Strategic Capabilities Required

- [ ] **Idempotency:**
  - Activities must be safe to retry—same result executing multiple times
  - Database operations use unique constraints to prevent duplicates
  - External API calls include idempotency tokens for deduplication

### Failure Detection:

- Long-running activities report progress via heartbeats
- Failures detected in minutes, not hours
- Retry policies aligned with actual failure characteristics

### Multi-Step Transactions:

- Saga pattern implemented for operations spanning multiple services
- Explicit compensation logic when partial failures occur
- Rollback procedures tested, not just designed

## Success Criteria:

- **Idempotency validated**—concurrent retry scenarios cause no duplicate side effects
- **Heartbeat mechanisms implemented**—long-running activities signal progress
- **Retry policies calibrated**—based on real-world service behavior, not defaults
- **Compensation logic tested**—rollback procedures work under failure injection

## Check Red Flags

**Red Flag :01**

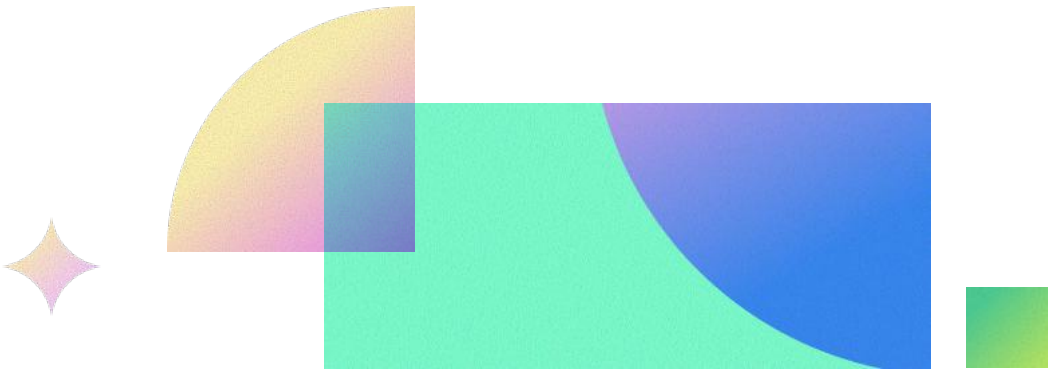
We'll add idempotency if we see duplicate transactions" (reactive, not proactive)

**Red Flag :02**

Using default retry policies without understanding failure characteristics

**Red Flag :03**

No testing of error paths or compensation logic



# 6. Workflow Versioning: Deploy Without Breaking Production

### The Business Question:

- [ ] Can we deploy code changes without disrupting workflows already in progress?

### The Risk:

- [ ] Simple optimization deployed. 147 in-flight orders fail with "non-deterministic error."
- [ ] Customers 90% through checkout lose carts.
- [ ] Order completion drops 23%. Rollback is quick.
- [ ] Damage to customer orders is permanent.

### The Strategic Capability Required

**Code Evolution Without Disruption:**

- [ ] In-flight workflows complete using original logic
- [ ] New workflows execute with updated code
- [ ] No maintenance windows or migration scripts required
- [ ] Friday deployments become routine, not risky

### The Implementation Approaches

- [ ] **GetVersion API:** Gradual migration—old and new code paths coexist safely

- [ ] **Worker-Based Versioning:** Separate worker fleets for major architectural changes

### Success Criteria:

- [ ] **Versioning strategy selected**—appropriate for deployment frequency and change complexity
- [ ] **Backward compatibility verified**—in-flight workflows tested against new code

- [ ] **Deployment procedures documented**—team knows how to evolve workflows safely
- [ ] **Rollback plan established**—clear procedure if version compatibility breaks

### Check Red Flags

- [ ] **Red Flag :01**
  We'll just deploy carefully and hope nothing breaks
- [ ] **Red Flag :02**
  No testing of new code against in-flight workflow scenarios
- [ ] **Red Flag :03**
  Treating workflow code like stateless API services

## 7. Testing Strategy: Finding Bugs Before Customers Do

### The Business Question:

- [ ] Are we confident workflows behave correctly under all conditions, or are we discovering bugs in production?

### The Risk:

- [ ] Payment workflow runs perfectly for six months. Holiday weekend arrives. Workflows start timing out.
- [ ] Root cause: retry policy configured for 30 seconds when payment gateway needs 2 minutes under peak load.
- [ ] Lost revenue: $200,000. Fix duration: 10 minutes. The bug was preventable with a 30-second test.

### The Testing Capabilities Required

**Comprehensive Validation:**

- [ ] Workflow logic verified in isolation from external dependencies
- [ ] Error handling paths tested, not just happy paths
- [ ] Timeout configurations validated against realistic service behavior
- [ ] Time-based behavior tested without waiting hours

**Continuous Validation:**

- [ ] Tests run automatically on every code change
- [ ] Breaking changes detected before reaching production
- [ ] Coverage targets enforce testing discipline

### Success Criteria:

- [ ] **TestWorkflowEnvironment integrated**—in-memory testing framework operational
- [ ] **Error path coverage achieved**—failure scenarios tested, not just success cases
- [ ] **Timeout validation completed**—configurations match real-world service behavior
- [ ] **CI/CD pipeline established**—automated testing prevents regressions

### Check Red Flags

- [ ] **Red Flag :01**
  We tested the happy path, that's enough.
- [ ] **Red Flag :02**
  No automated testing—relying on manual QA processes
- [ ] **Red Flag :03**
  Discovering timeout misconfigurations during peak traffic

## 8. Operational Excellence: The Details That Compound

### The Business Question:

- [ ] Have we addressed the operational details that seem minor but compound into incidents?

### The Risk:

- [ ] Workflows run perfectly for six months. Performance suddenly degrades. Event history queries timeout.
- [ ] The culprit: workflows passing entire JSON documents instead of external storage references.
- [ ] Storage bloat accumulates. Performance collapses. The pattern was preventable with proper guidance.

## The Operational Patterns Required

### Data Management:

- [ ] Large payloads stored externally (S3, databases), not in event history
- [ ] Lightweight references passed through workflows
- [ ] Storage growth monitored and bounded

### Operational Visibility:

- [ ] Metrics exported to standard monitoring infrastructure
- [ ] Dashboards focused on operational scenarios, not vanity metrics
- [ ] Search attributes defined for business-relevant queries

### Network Configuration:

- [ ] Corporate proxy compatibility validated before production
- [ ] Long-lived gRPC connections tested through infrastructure
- [ ] Workers secured behind appropriate network boundaries

## Success Criteria:

- [ ] **Payload patterns established**—team knows when to use external storage
- [ ] **Metrics infrastructure operational**—Prometheus, Grafana, alerts configure
- [ ] **Search schema defined**—operational queries work for business needs
- [ ] **Network integration validated**—proxy configuration tested end-to-end

## Check Red Flags

- [ ] **Red Flag :01**
  No guidance on payload size management—teams discover limits through incidents
- [ ] **Red Flag :02**
  Metrics infrastructure "coming soon"—flying blind initially
- [ ] **Red Flag :03**
  Network configuration untested until production deployment

# Strategic Deployment Authorization: The Final Assessment

Before authorizing production deployment, answer these questions honestly:

### ☐ Infrastructure Strategy:

- Build vs. buy decision made with clear business rationale
- Total cost of ownership calculated including engineering time
- Operational capacity assessed realistically, not optimistically

### ☐ Capacity & Scalability:

- Autoscaling configured based on meaningful capacity indicators
- Load testing completed simulating realistic traffic patterns
- Capacity headroom verified for unexpected growth

### ☐ Security & Compliance:

- End-to-end encryption validated—no plaintext exposure
- Secure debugging enabled—engineers can troubleshoot safely
- Compliance review completed—regulatory requirements satisfied

### ☐ Observability & Debugging:

- Event history replay capability validated and team trained
- Metrics pipeline operational with appropriate alert thresholds
- Operational queries work for business-relevant criteria

### ☐ Reliability & Fault Tolerance:

- Idempotency validated—retries cause no duplicate side effects
- Failure detection mechanisms implemented and tested
- Compensation logic validated for multi-step transactions

### ☐ Versioning & Code Evolution:

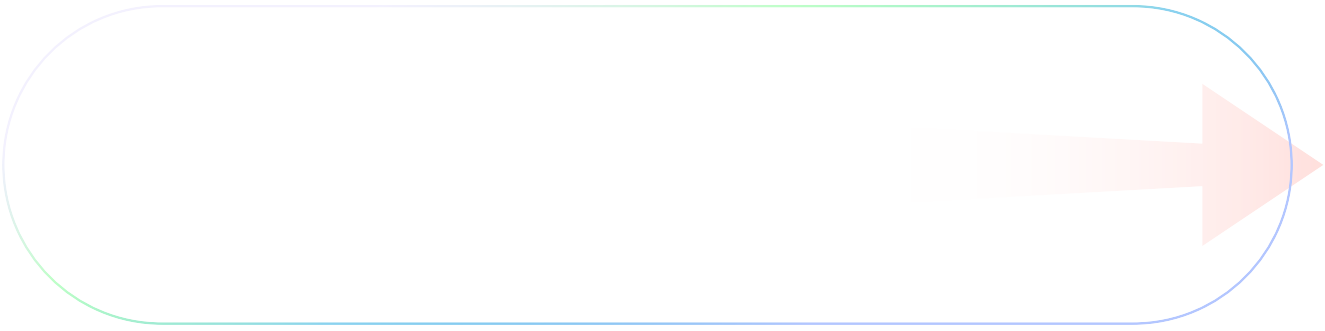
- Versioning strategy selected and deployment procedures documented
- Backward compatibility verified for in-flight workflows
- Team trained on safe code evolution practices

### ☐ Testing & Validation:

- Comprehensive testing framework operational
- Error paths tested, not just happy paths
- CI/CD pipeline prevents regressions

### ☐ Operational Excellence:

- Payload management patterns established and documented
- Monitoring infrastructure operational before first deployment
- Network integration validated end-to-end



# The Strategic Question

The question isn't whether Temporal can be deployed to production.

The question is whether the deployment is executed once, correctly—or rushed and followed by a year of architectural remediation.

Organizations that apply this framework succeed on their first production rollout. They avoid compliance failures, prevent performance degradation, and keep engineering capacity focused on product and revenue-driving initiatives instead of operational recovery.

Organizations that skip these decisions discover the consequences through costly production incidents and reactive rebuilds.

The difference between these outcomes is **execution discipline**, not chance.

# Deployment Authorization

Based on this assessment, production deployment is:

- ☐ **APPROVED** - All critical criteria satisfied, risks understood and accepted
- ☐ **CONDITIONAL** - Approved with specific mitigations documented below
- ☐ **DEFERRED** - Critical gaps identified, deployment postponed until resolved

### Critical Gaps Requiring Resolution:

_____

_____

_____

_____

**Deployment Date:** _____

**Deployment Lead:** _____

**Approving Authority:** _____

**Signature:** _____