Promise Protocol v1 — The Guided Tour

Week 12 of 12: Drills, Polish & Buffer

Your Software Pedagogue

September 17, 2025

Contents

1 This Week's Mission

The application is built, deployed, and observable. But is it truly resilient? **Thinking** a system is resilient is not the same as **knowing** it is. This week, our mission is to gain that knowledge through practice. We will conduct "disaster drills" to test our assumptions and automated systems. We will write clear instructions—runbooks—for future human operators. And finally, we will use this week as a buffer to polish our work and ensure we finish with a system that is not only complete but truly production-ready.

2 Learning Plan

To prepare for our disaster drills, we need to know what kind of disasters to simulate. This week's reading provides a menu of common real-world failures.

2.1 TODO Reading Assignment

- ☐ Read **Chapter 24: Common failure causes** in *Understanding Distributed Systems.*
  - **Focus On**:
    - [cite_start]The different categories of faults: hardware faults, configuration changes, network faults, resource leaks, and especially **cascading failures**[cite: 2833, 2847, 2869, 2882, 2908].
    - [cite_start]The concept of "gray failures"[cite: 2877]: subtle, partial degradations that are hard to detect.
    - [cite_start]The idea that incorrect error handling is a primary cause of major outages[cite: 2842].
  - *Why this now?*: This chapter is your playbook for this week's main activity: Chaos Engineering. It gives you a list of realistic failure scenarios to inject into your system to see if it responds as you expect.

3 Building Plan: A Step-by-Step Guide

This week is about validation and preparation, not writing new feature code.

3.1 TODO Step 12.1: Write the Team Runbooks

When an alert fires at 3 AM, the on-call engineer shouldn't be debugging from scratch. They need a clear, calm checklist to follow. That checklist is a runbook.

- **Key Concept: Runbook**. It's like an **airplane pilot's emergency checklist**. When an engine fails, the pilot doesn't rely on memory; they pull out a laminated checklist and follow the steps precisely. A runbook is that checklist for a specific production alert.

- ☐ For each critical alert you configured in Week 11 (e.g., "High API Error Rate," "High Payment Latency"), create a corresponding runbook document.

- ☐ Each runbook should contain:
  - A summary of what the alert means.

  - The immediate first steps to take (e.g., "Check the status of the PSP," "Look at the main service dashboard").
  - Links to the relevant dashboards.
  - Queries to run against your logs or traces to find more information.
  - Known mitigation steps (e.g., "If this is caused by a bad deployment, initiate a rollback using this procedure. . . ").

3.2 TODO Step 12.2: Conduct Disaster Drills (Chaos Engineering)

Now, we test our system and our runbooks. We will intentionally inject failures into our production-like staging environment to see what happens.

- **Key Concept: Chaos Engineering**. This is like a **fire drill for your application**. You don't wait for a real fire to find out if the alarm works and if people know the escape routes. You practice. Chaos Engineering is the disciplined practice of experimenting on a system in order to build confidence in its ability to withstand turbulent conditions.

☐ Schedule a "Game Day" with a clear plan of what you will test. Announce it so no one is surprised.

☐ **Drill 1: Downstream Dependency Outage**

  - -[ ] Simulate the PSP being unavailable (e.g., by adding a network rule to block traffic to the Stripe API).
  - -[ ] **Expected Outcome**: The circuit breaker on your PSP gateway should trip. The "High Payment Latency" alert should fire. Your runbook for this alert should be useful. The system should gracefully degrade without crashing.

☐ **Drill 2: Database Failover**

  - -[ ] Simulate a database replica failure (e.g., by manually killing the primary database pod in Kubernetes).
  - -[ ] **Expected Outcome**: Your managed database service should automatically fail over to a healthy replica. Your application may see a brief spike in errors but should recover automatically without manual intervention.

- ☐ **Drill 3: Cascading Failure / Resource Leak**
  - -[ ] Simulate a "poison pill" request that causes one of your API pods to enter a high-CPU or high-memory state.
  - -[ ] **Expected Outcome**: The pod's health check should eventually fail. Kubernetes should kill the unhealthy pod and start a new, healthy one. The load balancer should have routed traffic away from the failing pod, causing minimal impact to users.

3.3 TODO Step 12.3: Final Polish & Buffer Time

No project plan is perfect. This final set of tasks is for handling the loose ends and absorbing any delays from the previous 11 weeks.

- ☐ Address any bugs, flaky tests, or weaknesses you discovered during the disaster drills.
- ☐ Update your runbooks with the lessons you learned from the drills.
- ☐ Review and finalize all project documentation (e.g., your READMEs, architectural decision records).
- ☐ Clean up any remaining technical debt.
- ☐ Use this time to catch up on any tasks from previous weeks that may have spilled over.
- ☐ **Congratulations!** At the end of this week, you have a feature-complete, well-documented, and battle-tested system.