Chaos Testing APIs with Fault Injection and Network Partitioning

🧪 What is Chaos Testing?

Chaos testing is a discipline within **resilience engineering** that involves deliberately introducing failures into a system to verify its ability to withstand turbulent conditions in production. When applied to **APIs**, chaos testing focuses on how the services behave under various failures such as timeouts, dropped connections, latency spikes, and backend outages.

🔧 Techniques for Chaos Testing APIs

1. Fault Injection

Fault injection introduces artificial errors in a controlled environment to simulate real-world failures.

Types of Faults:

Fault Type	Example
Latency	Add delay to API response (e.g., +500ms)
Error response	Force 500/503 errors from dependent APIs
Resource limits	Simulate CPU/memory exhaustion
DNS failures	Fail hostname resolution for API targets
Connection drops	Kill TCP connections randomly

Tools:

- **Gremlin**
- **LitmusChaos**

- **Toxiproxy**
- **Chaos Mesh**
- [**Istio Fault Injection** (Service Mesh)]

⚙️ Injecting Chaos in a CI/CD Pipeline

Chaos tests can be automated post-deployment:

```
stages:
  - deploy
  - chaos
  - validate

chaos:
  script:
    - chaosctl inject fault --target=auth-service --latency=500ms
    - chaosctl inject partition --from=api-service --to=db-service
  only:
    - staging
```

You can validate via health checks, alert logs, and synthetic monitoring during the chaos test window.

🧠 Best Practices

- Always monitor **metrics and logs** during chaos experiments
- Define clear **SLOs/SLAs** and **recovery thresholds**
- Use **feature flags** to disable chaos in critical paths
- Run chaos tests in **non-production** first
- Always run **post-chaos assertions** to validate recovery

✅ Summary

Chaos testing APIs helps simulate unexpected behavior and infrastructure failures in order to build **resilient systems**. With fault injection and network partitioning techniques, you can proactively identify failure points and harden your APIs before real outages occur.