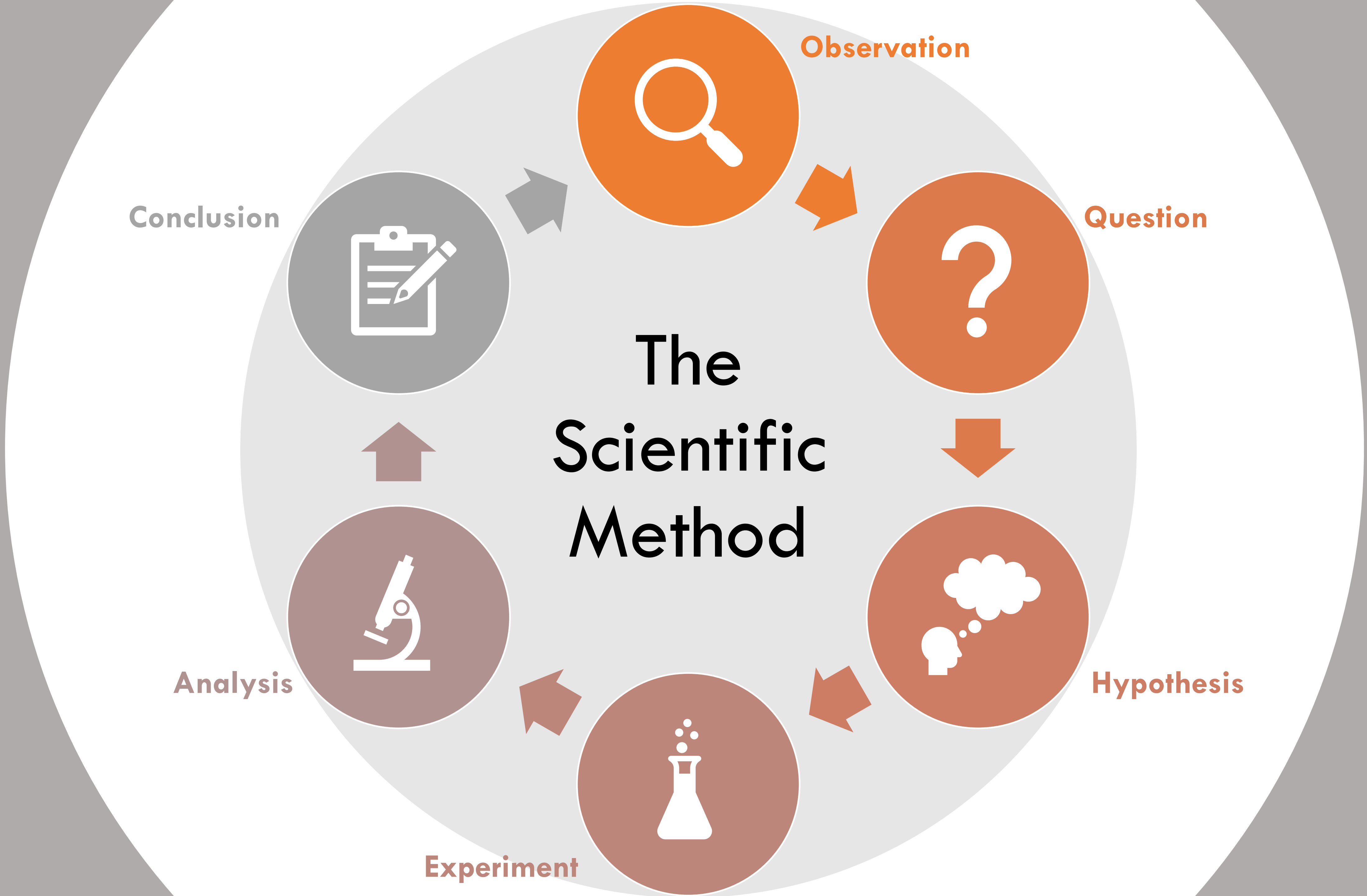
The Scientific Method for Resilience

Christina Yakomin

Vanguard


Observation
Question
Hypothesis
Experiment
Analysis
Conclusion
The Scientific Method

How does this apply to **resilience?**

1. Failure Modes and Effects Analysis
2. Chaos Engineering
3. Documentation & Planning

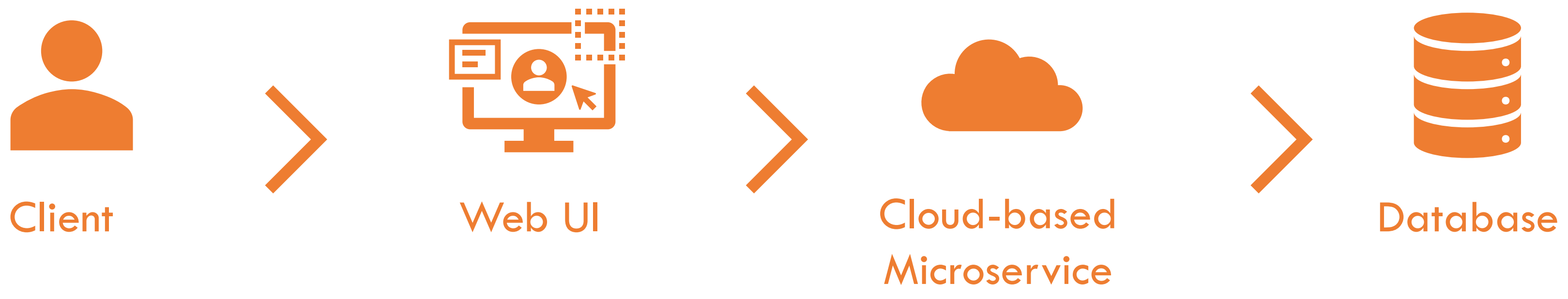
Monday	Tuesday	Wednesday	Thursday	Friday
2pm - Failure Modes and Effects Analysis	Task due – List of Hypotheses	Prepare for experimentation	11am – Chaos GameDay Activity	Task due – Publish findings

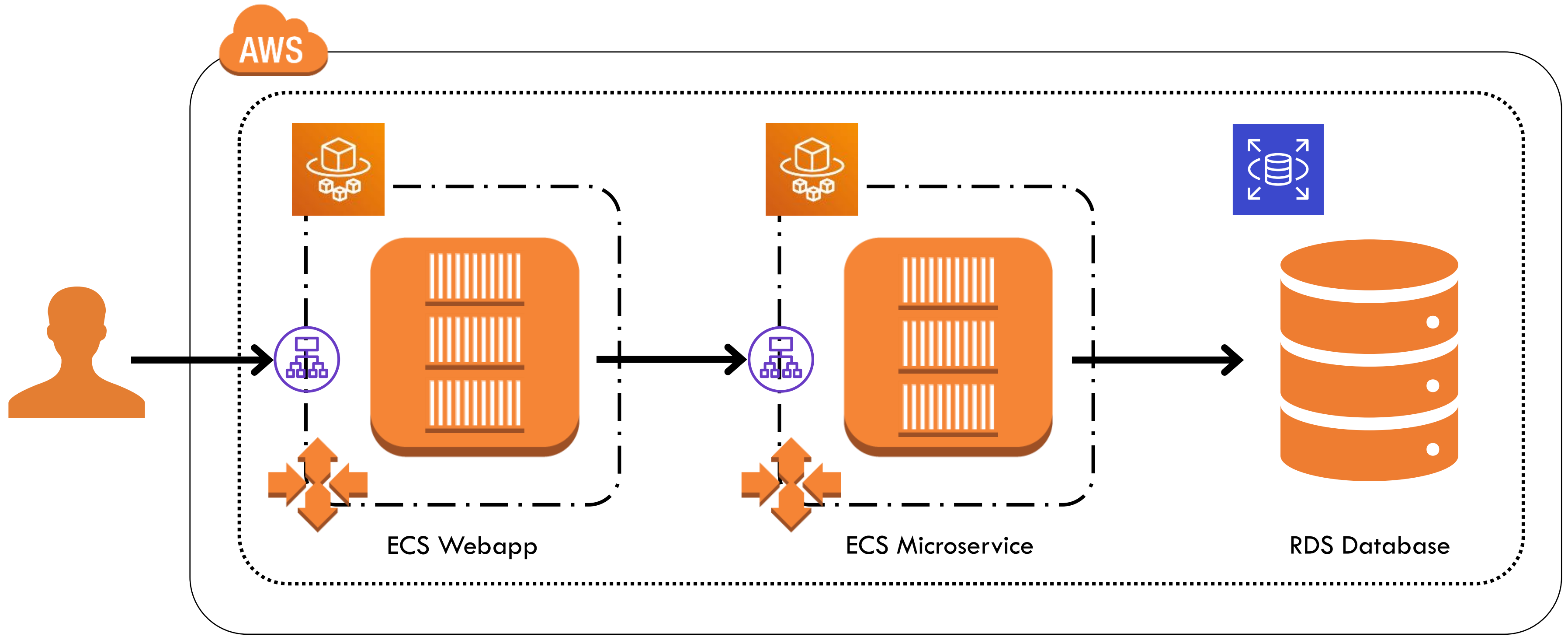
Scenario: our schedule for the hypothetical week

Step 1: Observation

- Reference an architecture diagram
- Identify critical components
- Consider the business process flow

“Here’s our sample system architecture! Let’s discuss how resilient it is.”



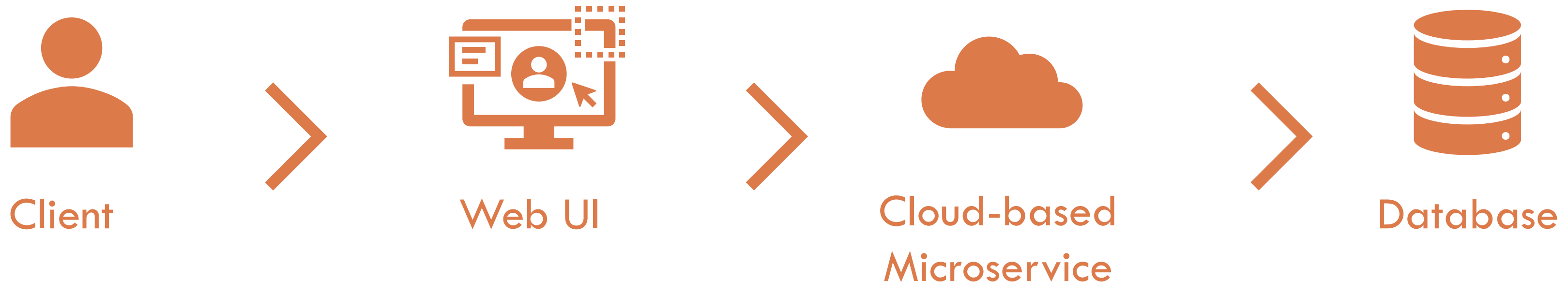


Our Simple Sample Architecture

Step 2: Question

- Discuss how each might component fail
- What would the effect be in each of the failure scenarios?

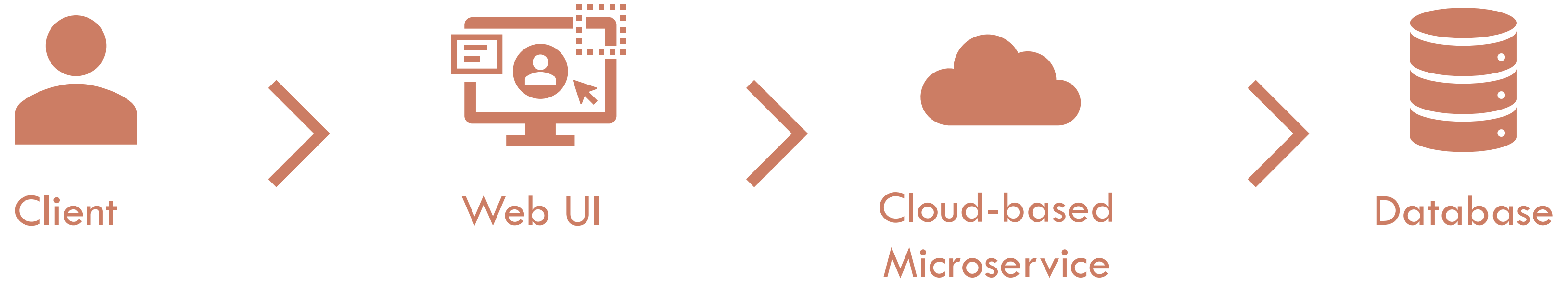
"What do you think would happen if our database became unavailable?"



Step 3: Hypothesis

- Based on what the team knows about the system, discuss the answers to these questions
- Develop a hypothesis based on the group consensus
- People may not always agree!

“If our database became unavailable, writes would fail, but reads would be served from our microservice’s in-memory cache.”

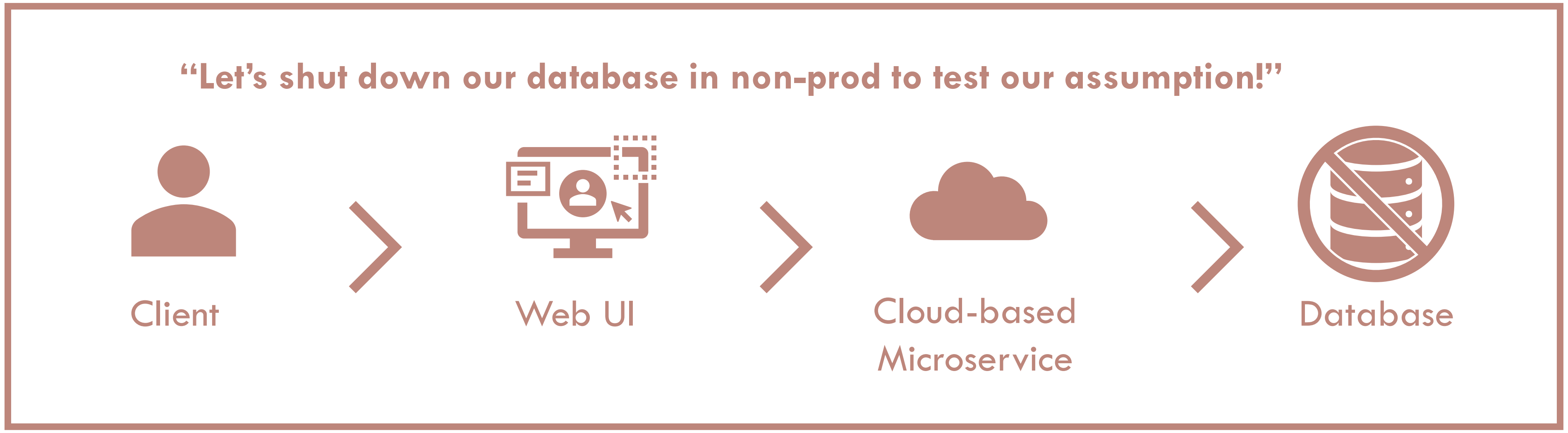


Failure Modes and Effects Analysis

Process Step	Failure Mode	Expected Behavior	Hypothesis
Web UI sends request to Microservice to read info from database	Microservice is unavailable or returns an error	Respond to Web UI with an error indicating downtime	If the microservice is unavailable, then reads will fail
Microservice tries to read info from database	Database is unavailable or returns an error	Send back response with cached data from in-memory cache	If the database is unavailable, **then reads will continue to succeed for a while** due to the in-memory cached data
Web UI sends request to Microservice to write update to database	Microservice is unavailable or returns an error	Respond to Web UI with an error indicating downtime	If the microservice is unavailable, then writes will fail
Microservice tries to write update to database	Database is unavailable or returns an error	Respond to Web UI with an error indicating downtime	If the database is unavailable, then writes will fail

Step 4: Experiment

- Run a test! Whether you're using a vendor tool, an open source library, homegrown automation, or manual steps – inject the failure mode into the system.


"Let's shut down our database in non-prod to test our assumption!"
Client
Web UI
Cloud-based Microservice
Database

Mechanisms for Fault Injection

Advanced Chaos Tools

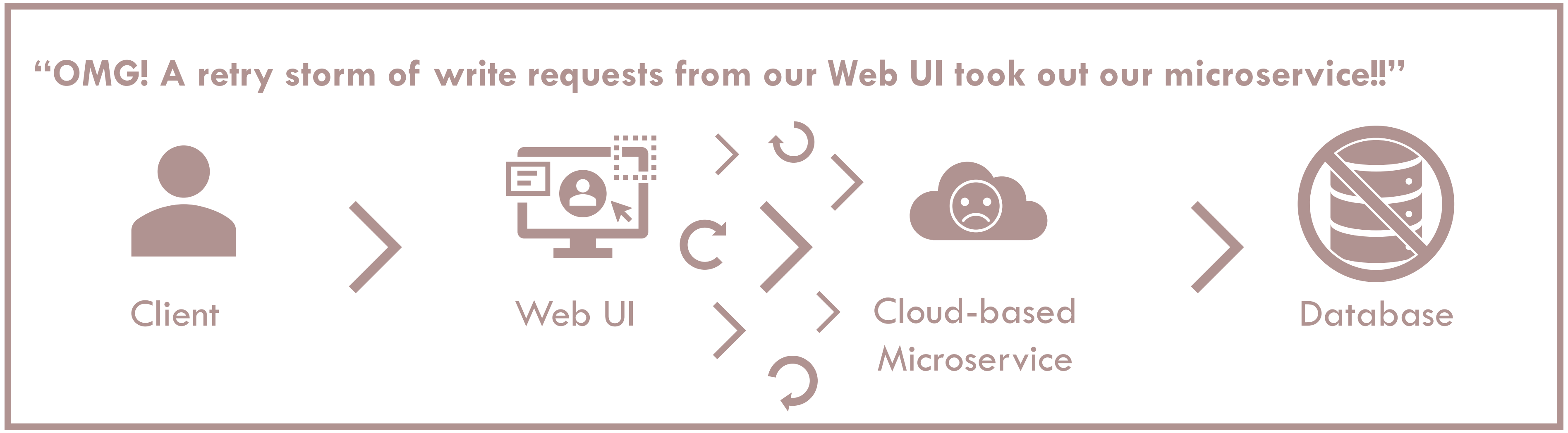
Open Source Libraries

Custom Scripts

Manual Efforts

Step 5: Analysis

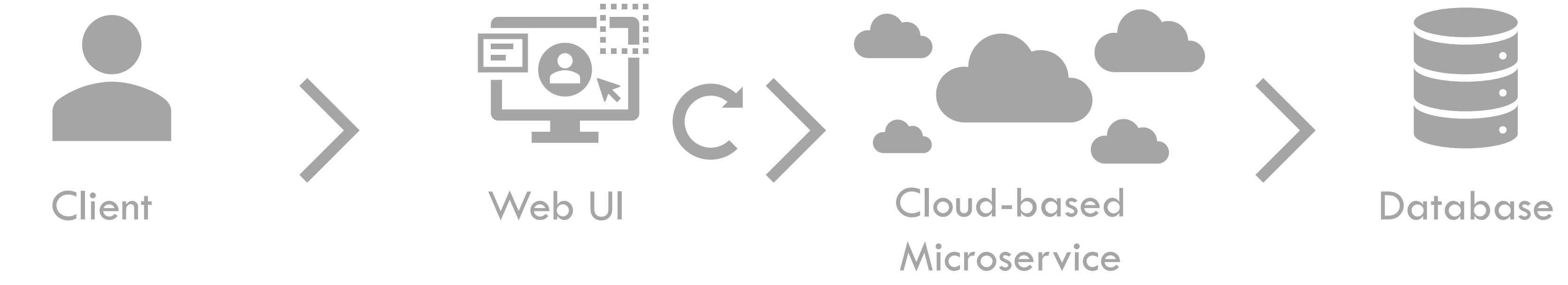
- Use the available Telemetry/Observability to see the effects of the injected fault
- Compare these observations to the hypotheses. Were the team’s expectations met?


“OMG! A retry storm of write requests from our Web UI took out our microservice!!”
Client
Web UI
Cloud-based Microservice
Database

Step 6: Conclusion

- Document your work! Make sure all of the steps are written down and observations have been captured
- Spend some time action planning
- Modify “variables” (make system changes) and repeat!

“Let’s implement a circuit breaker in our Web UI, and better retry logic in our microservice so we’re more resilient to database failures. Then we’ll re-test!”



Process Step	Failure Mode	Actual Behavior	Desired Behavior	Remediation Plan
Web UI sends request to Microservice to read info from database	Microservice is unavailable or returns an error	Web UI will retry forever with no limits	Use a circuit breaker to fail fast without overloading the microservice	Implement the circuit breaker pattern around the microservice request
Microservice tries to read info from database	Database is unavailable or returns an error	Send back response with cached data from in-memory cache	Send back response with cached data from in-memory cache	No action required
Web UI sends request to Microservice to write update to database	Microservice is unavailable or returns an error	Web UI will retry forever with no limits	Use a circuit breaker to fail fast without overloading the microservice	Implement the circuit breaker pattern around the microservice request
Microservice tries to write update to database	Database is unavailable or returns an error	Respond to Web UI with an error indicating downtime	Use limited retries with exponential backoff to handle transient database failures	Implement the retry logic around the database request

Vanguard's **real** stories

- Our first FMEA
- Critical Workflows
- FMEA at Scale

Christina Yakomin

Senior Technical Specialist
Site Reliability Engineering at Vanguard

/in/christina-yakomin

@SREChristina

Cloudy with a Chance of Chaos
SRECon ‘20