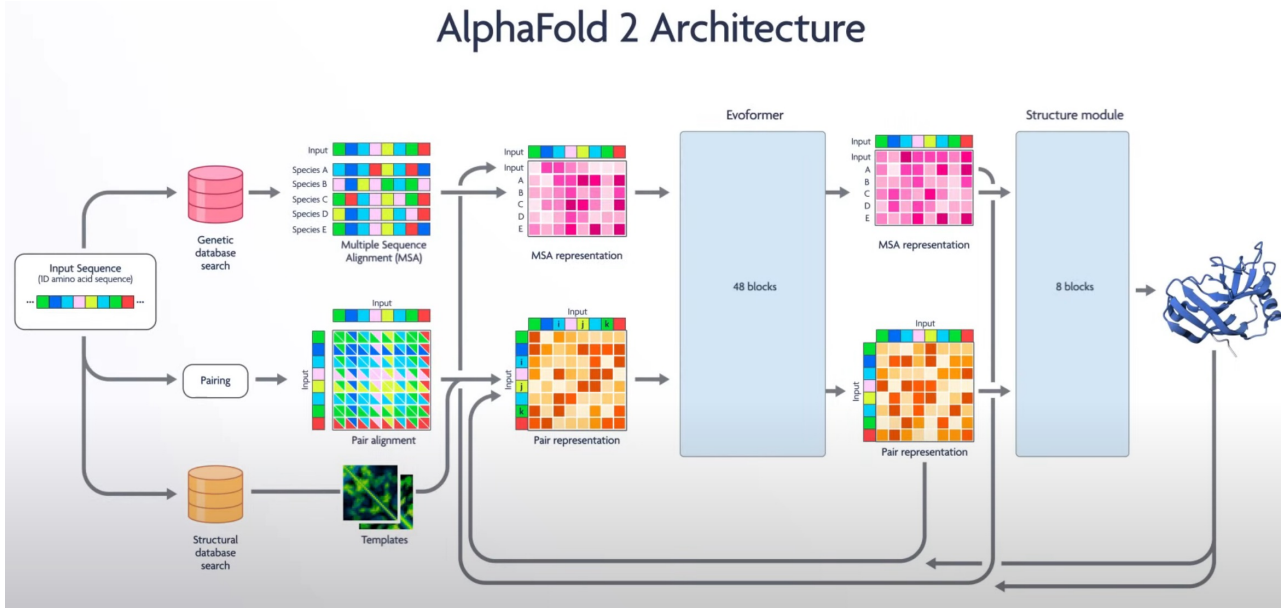

expo QA 24
MADRID
May 28th,
29th, 30th
2024
sogeti
expo:QA
expoqa.com

Diogo Goncalves Candeias

QA Unit Director | Head of Quality Assurance
GenAI, Big Data & Digital Assurance enthusiast @Exceltic

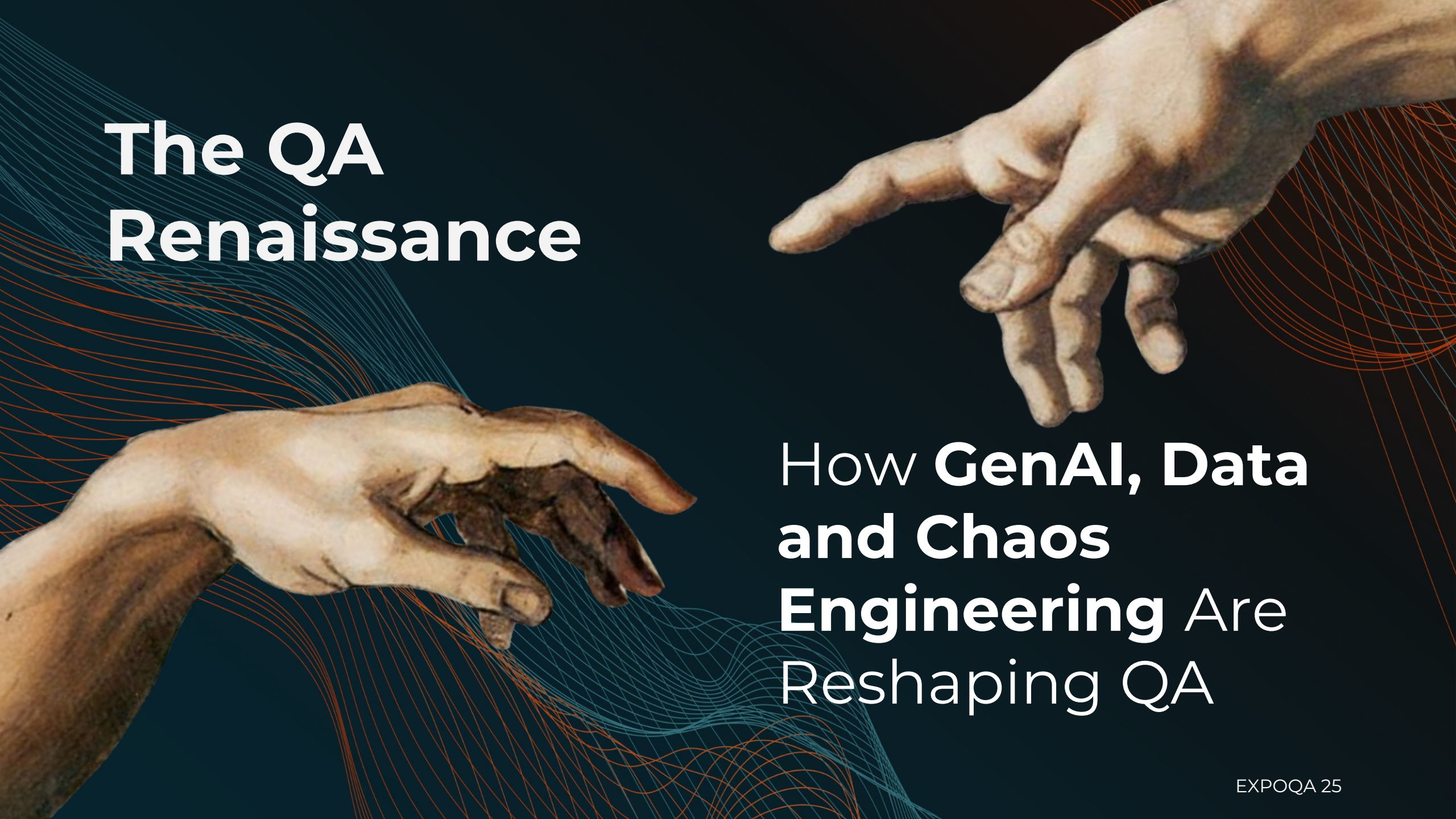


The QA Renaissance

How **GenAI, Data and Chaos Engineering** Are Reshaping QA

EXPOQA 25

The QA Renaissance



How **GenAI, Data and Chaos Engineering** Are Reshaping QA

EXPOQA 25

1. **The Shift from QA to Quality Engineering**
2. **Generative AI:** LLMs & Test Generation
3. **Synthetic Data:** Realistic Test Data at Scale
4. **AI-Powered Observability:** Predictive Monitoring
5. **Intelligent Chaos Engineering:** Dynamic Resilience Testing
6. **Wrap-up + Q&A**

From QA to Quality Engineering

AI, Data, and Chaos at the Helm of Software Quality

Late Bug Detection: Classic QA finds defects at end of cycle, causing costly fixes

Slow & Labor-Intensive: Manual test case design and execution can't keep up with Agile speed

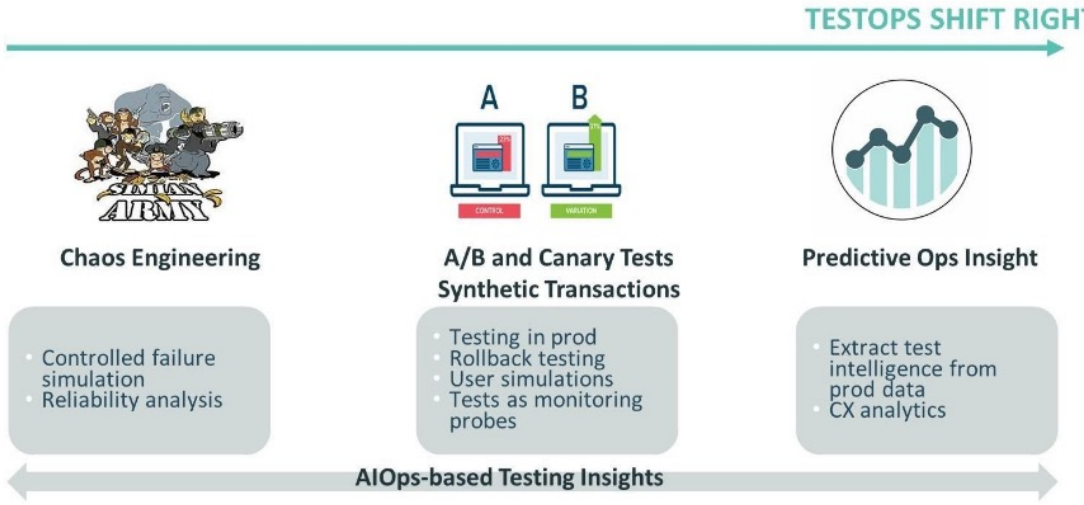
Limited Coverage: Hard to simulate all user scenarios or rare edge cases with hand-crafted tests

Reactive, Not Proactive: Focus on finding bugs post-development, rather than preventing them early

AI and automation technologies are key enablers of QE.
Automation reduces manual effort, and AI takes it further by intelligently generating test cases, analyzing results, and doing things traditional scripts can't.

QE also involves Shift-Right practices – testing in production or late-stage environments. This includes robust monitoring (observability) and chaos engineering to ensure the system behaves under real-world stresses.

Rather than assuming our pre-release tests caught everything, we continue to validate and learn from the system's behavior in the field (while minimizing user impact).



GenAI & QA

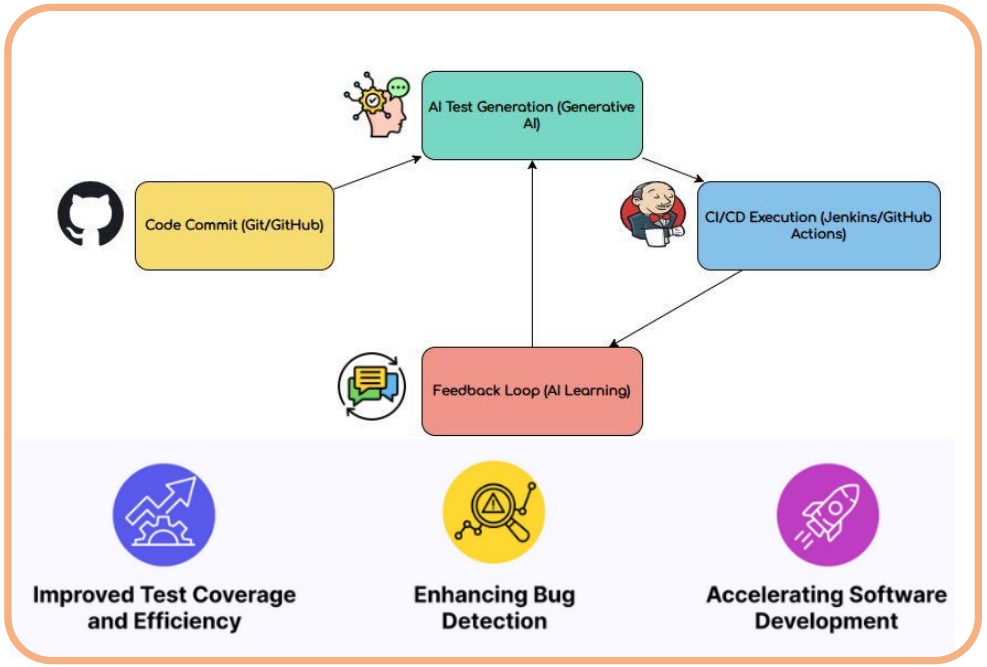
Generate tests from user stories or requirements and integrate them into frameworks.

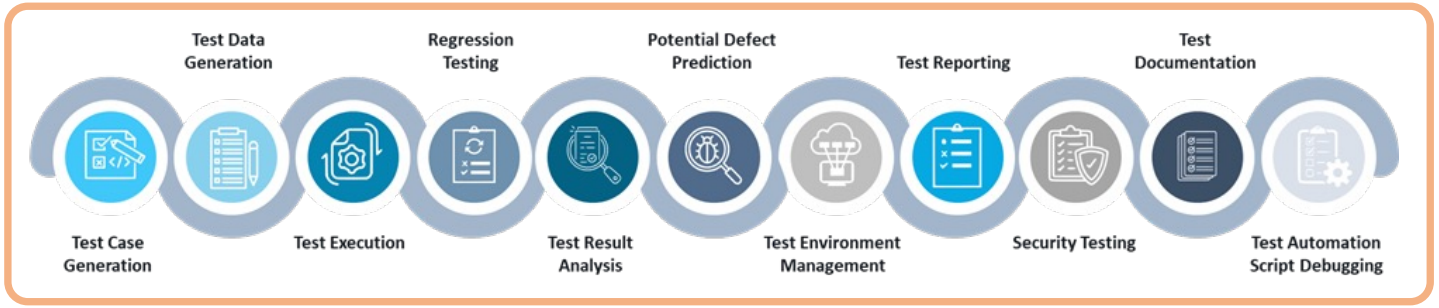
Generative AI in QA

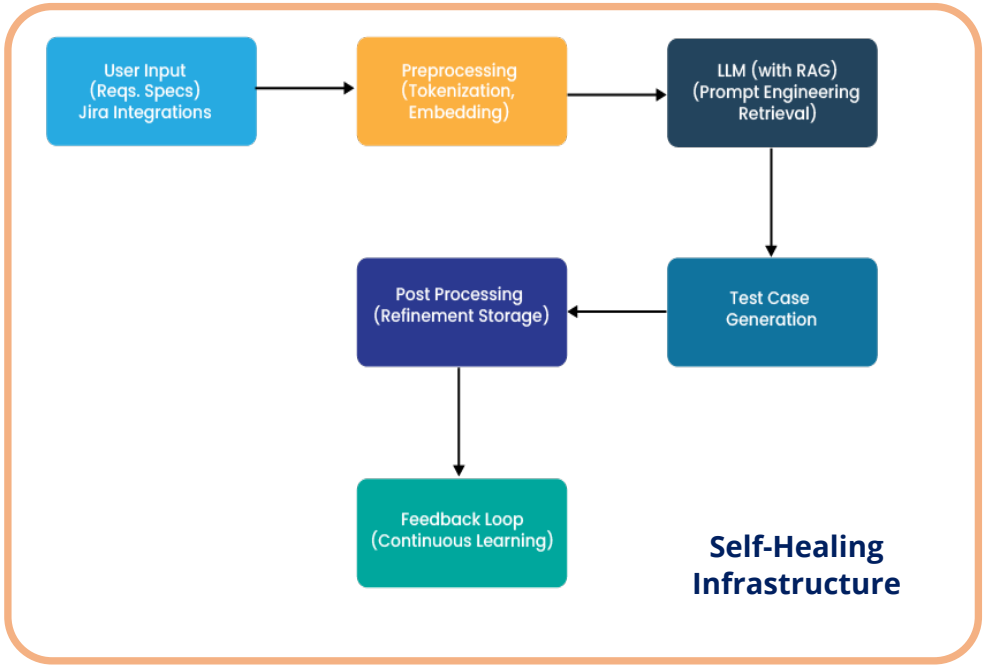
AI, Data, and Chaos at the Helm of Software Quality

One of the most promising use cases is automating the creation of **test artifacts**.

Think of all the test cases, user scenarios, and test scripts we write – generative AI can help draft those.







Generative AI in QA

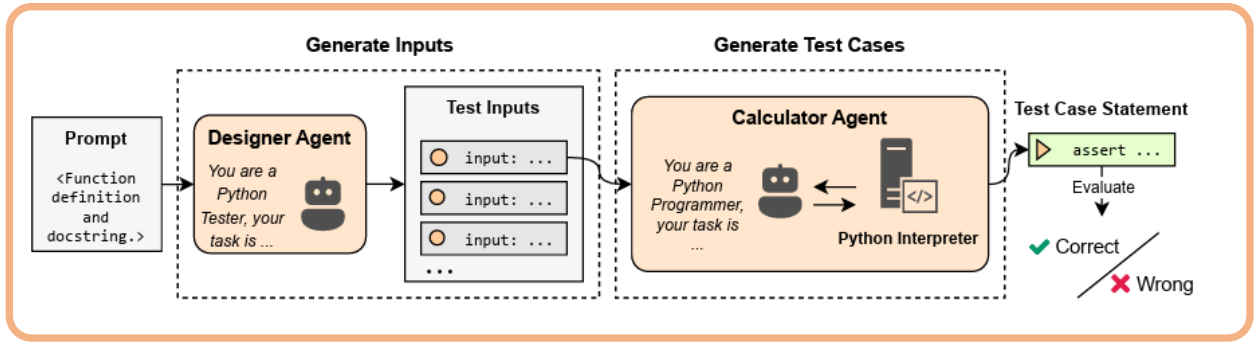
AI, Data, and Chaos at the Helm of Software Quality

Test Case Generation: Generate test scenarios from requirements or user stories (e.g. ChatGPT creating login test cases)

Test Script Automation: AI assistants (Copilot, Code LLMs) write Playwright/Appium scripts or unit tests from function names

Bug Detection & Triage: LLMs review code or logs to identify potential bugs, or explain failures in plain language

Documentation & Reports: Auto-generate test plans, summaries, and bug reports in clear language



```
feature_description = "Ensure that the shopping cart allows users to add items, remove items, and proceed to checkout."
regression_tests = generate_regression_tests(feature_description)
print(regression_tests)
```

```
Regression Test Scenarios for Shopping Cart Feature:

1. Test Case: Adding items to the shopping cart
   - Verify that the user can add a single item to the shopping cart.
   - Verify that the user can add multiple items to the shopping cart.
   - Verify that the quantity of the added items is correctly displayed in the shopping cart.
   - Verify that the total price of the added items is correctly calculated and updated in the shopping cart.
   - Verify that the user is able to add items with different variations (e.g., size, color).

2. Test Case: Removing items from the shopping cart
   - Verify that the user can remove a single item from the shopping cart.
   - Verify that the user can remove multiple items from the shopping cart.
   - Verify that the quantity and total price of the removed items are correctly updated in the shopping cart.
   - Verify that the user is able to remove items with different variations.

3. Test Case: Proceeding to checkout
   - Verify that the user can proceed to the checkout page from the shopping cart.
   - Verify that the user is redirected to the correct checkout page.
   - Verify that the items in the shopping cart are correctly displayed in the checkout page.
   - Verify that the total price of the items in the shopping cart is correctly displayed in the checkout page.
   - Verify that the user is able to navigate back to the shopping cart from the checkout page.
```

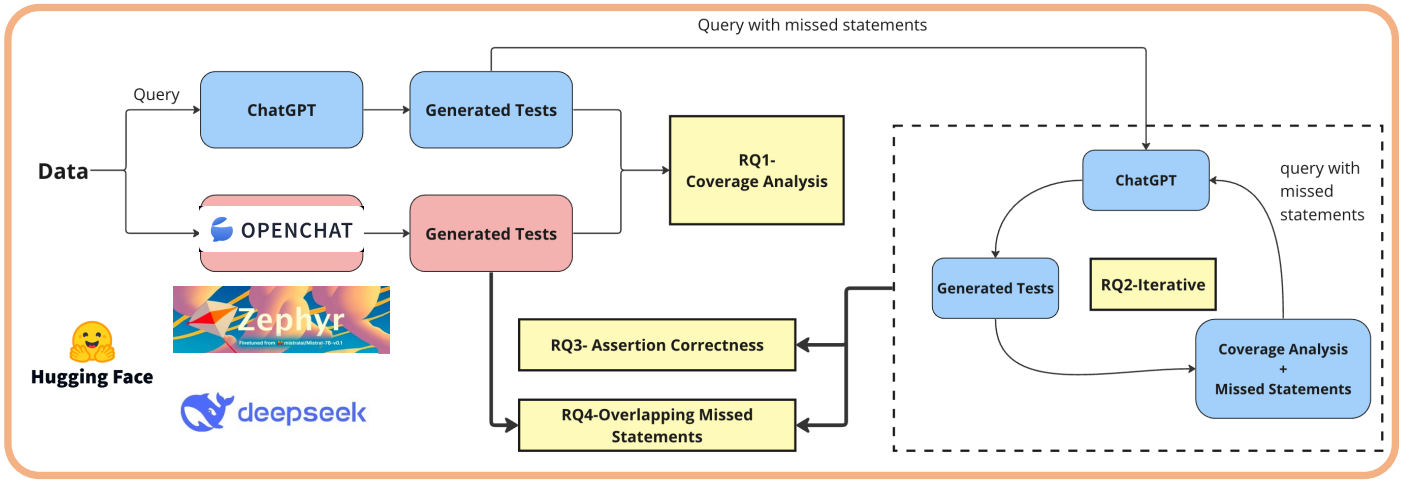
✓ 9s completed at 6:46 AM

```
api_description = "The weather API should return a JSON response with fields for temperature, humidity, and precipitation forecast for the next 5 days."
api_validation_tests = generate_api_validation_tests(api_description)
print(api_validation_tests)
```

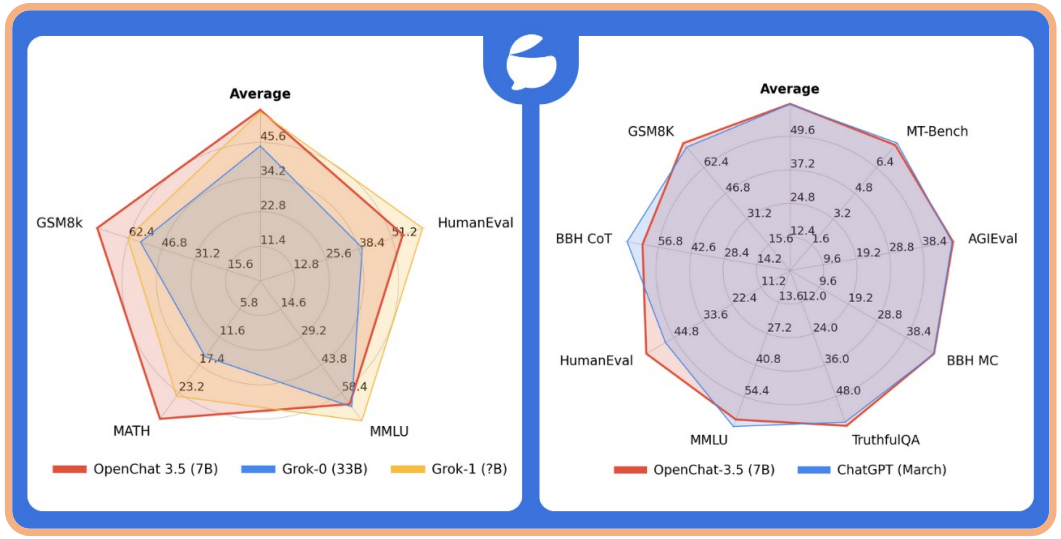
```
Here are some test cases to validate the JSON response from the weather API:

1. Test Case: Valid Response
- Description: Ensure that a valid JSON response is received with all the required fields (temperature, humidity, and precipitation) for the next 5 days.
- Expected Result: The response should contain the required fields and values for temperature, humidity, and precipitation for the next 5 days.

2. Test Case: Invalid Response Structure
- Description: Verify the response structure when the API returns an invalid JSON response.
- Expected Result: The response should not have any missing or unexpected fields and the structure should follow the defined schema.

3. Test Case: Missing Field - Temperature
- Description: Check the JSON response when the temperature field is missing for one of the days.
- Expected Result: The response should contain the temperature field for all 5 days. If the field is missing for any day, an error should be raised.

4. Test Case: Missing Field - Humidity
- Description: Check the JSON response when the humidity field is missing for one of the days.
- Expected Result: The response should contain the humidity field for all 5 days. If the field is missing for any day, an error should be raised.

5. Test Case: Missing Field - Precipitation
```

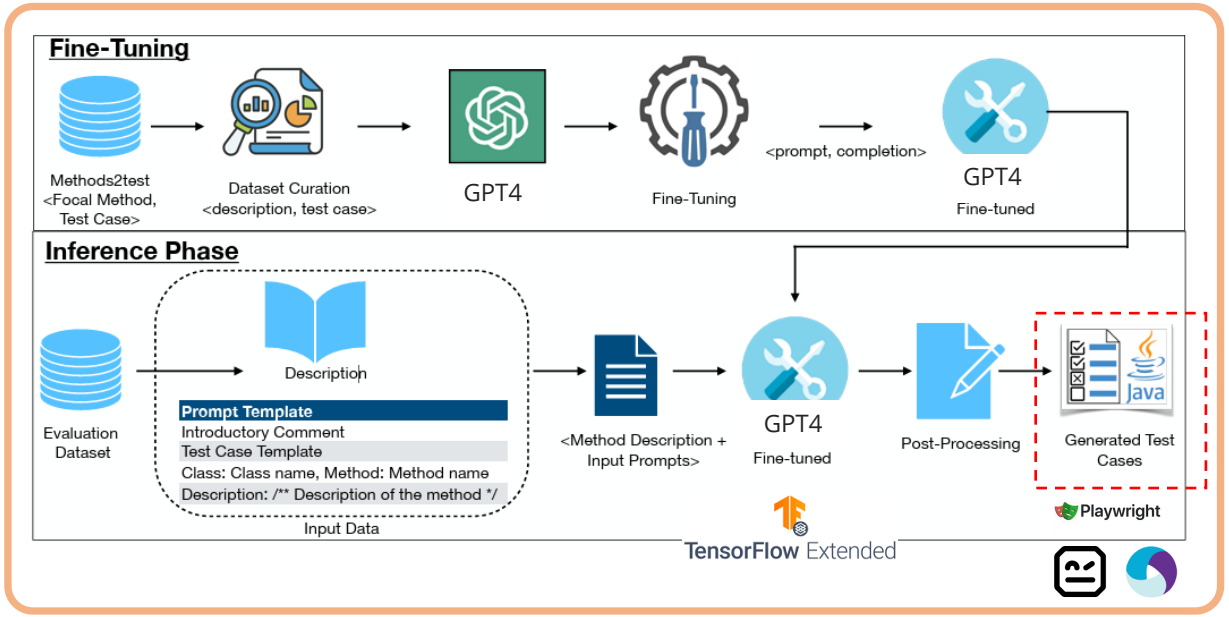
Demo – AI-Generated Test Cases

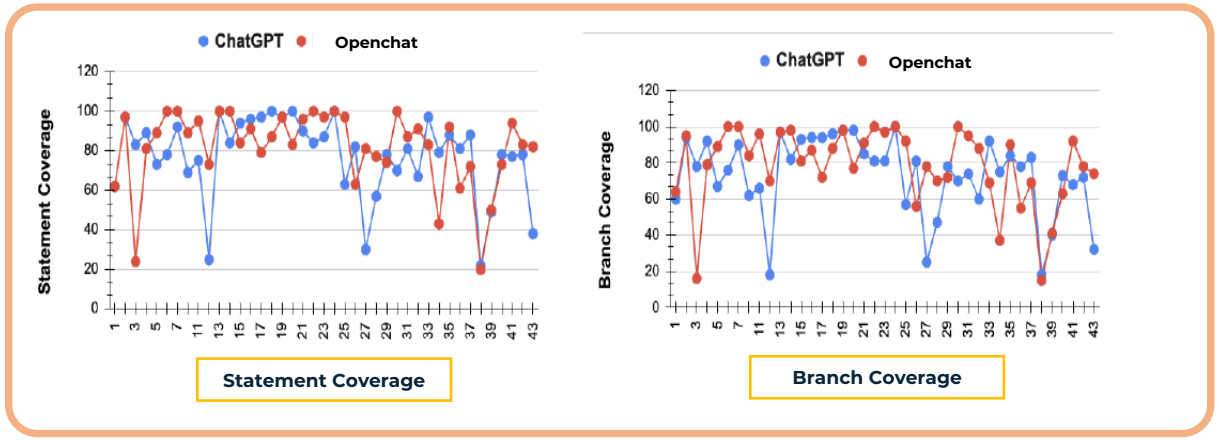
AI, Data, and Chaos at the Helm of Software Quality



0-100 LOC	Avg Statement Coverage ChatGPT	Openchat	Avg Branch Coverage ChatGPT	Openchat
Category 1 (original)	0	0	0	0
Category 1 (refactored)	97.45	0	96.85	0
Category 2	93.26	90.3	91.68	90.1
Category 3	91.55	97	89.5	96.15



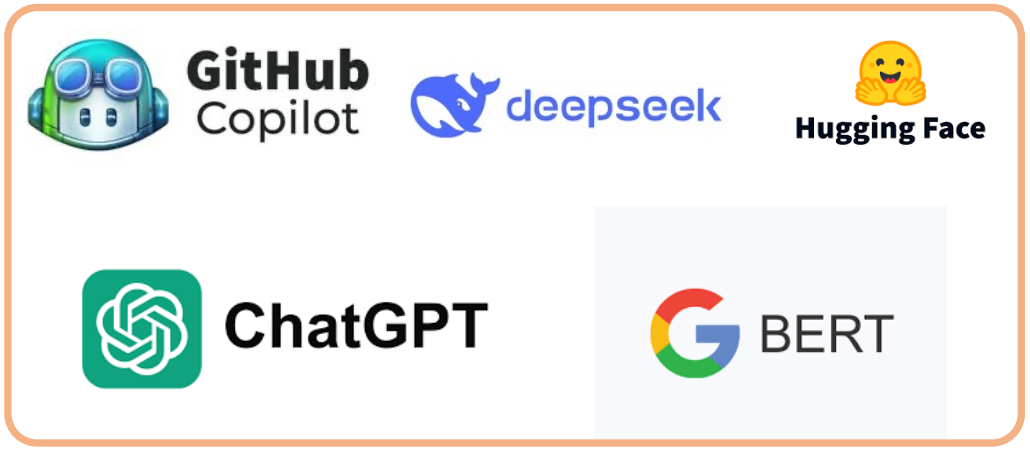




Applications of Generative AI in QA

AI, Data, and Chaos at the Helm of Software Quality

- **Speed & Efficiency:** Dramatically reduces test design and coding time – more coverage in less time
- **Improved Coverage:** AI suggests edge cases humans might miss, boosting test breadth
- **Consistency & Quality:** Uniform test case formats and thoroughness that's hard to achieve manually
- **Focus on Higher-Value Work:** Testers spend more time on strategy and analysis, less on rote writing
- **Predictive Analytics:** AI forecasts test flakiness before execution, rerunning only stable tests.
  Example: An ML model trained on past pipeline data predicts failures with 92% accuracy.
- **Self-Healing Infrastructure:** AI auto-rolls back deployments if performance thresholds are breached.
- **Auto-Documentation:** Tools like Swagger AI auto-generate test reports and audit trails for compliance.
- **Accuracy Validation:** AI-generated scripts undergo an accuracy check before deployment.



By **2026, AI will automate 80% of test maintenance tasks, freeing QA engineers** to focus on **exploratory testing and strategic quality advocacy.**

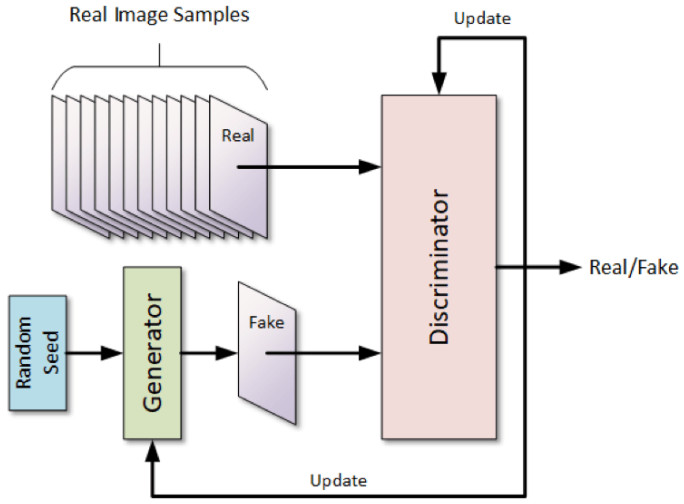
Synthetic Data

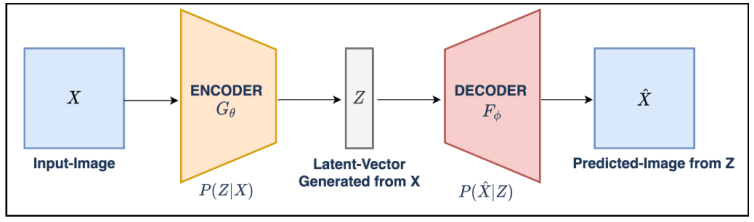
Create high-fidelity, privacy-preserving test datasets that mirror real-world behavior

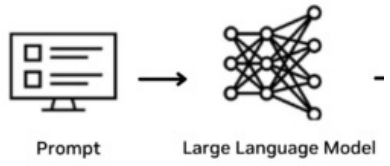
GenAI Models

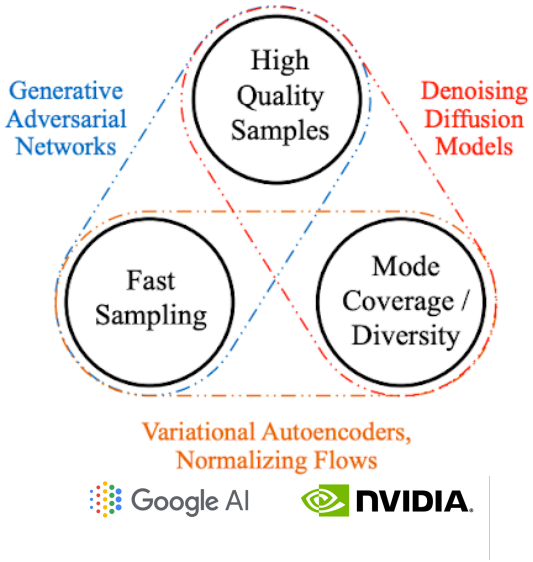
Generative Adversarial Network (GAN)

Unsupervised learning that utilize two neural networks. Pitted against each other, with one generating new data (such as images) that the second network then tries to identify as **real or generated**.









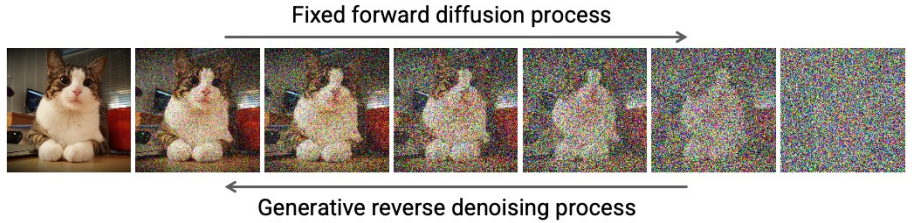
Variational Autoencoder (VAE)

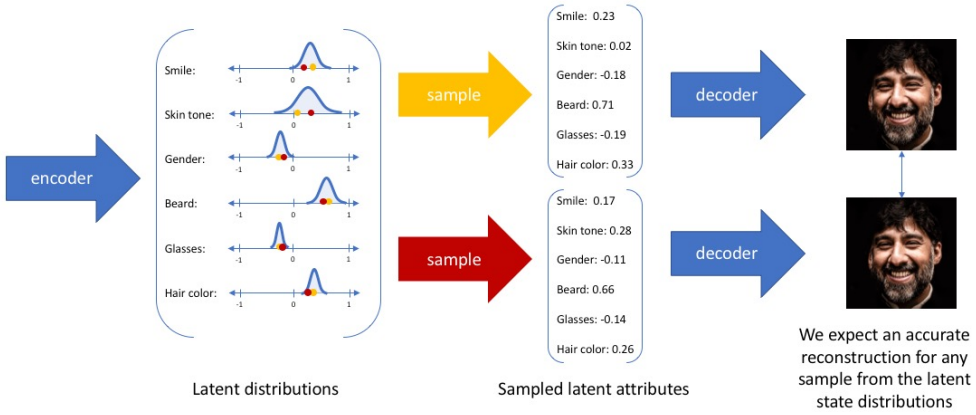
Type of neural network architecture that combines elements of both an autoencoder and a probabilistic model.

It is designed to learn a compressed representation of input data while also capturing the **underlying probability distribution of the data**.

Denoising Diffusion Models

Generative models that create new data samples by iteratively refining noisy versions of the data, utilizing a two-step process of **adding noise** (diffusion) and then **removing it** (denoising) to recover the original data.



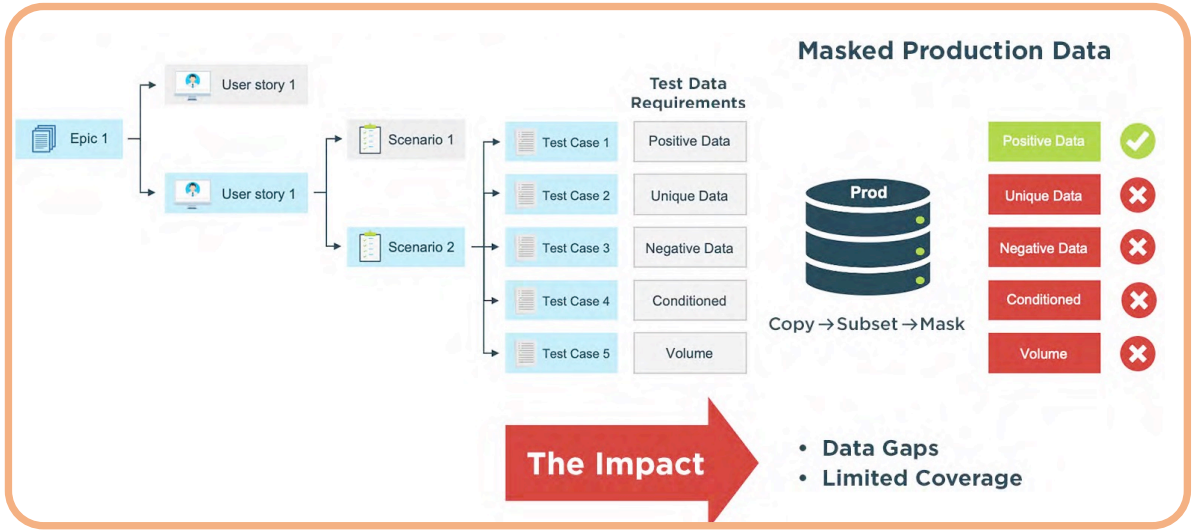


Intro for Synthetic Test Data

AI, Data, and Chaos at the Helm of Software Quality

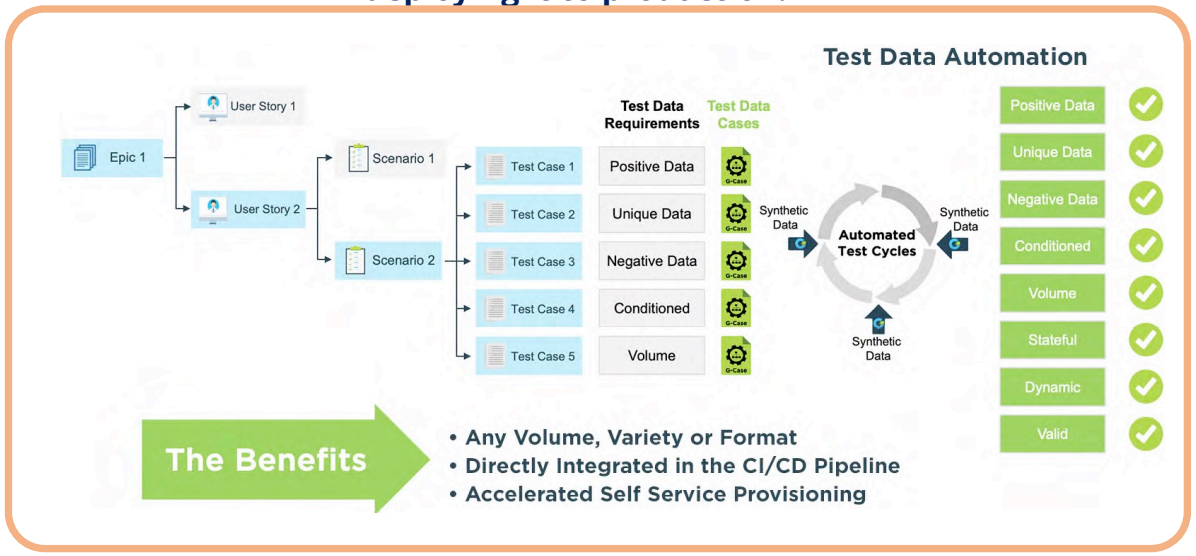
Synthetic test data is artificially generated data that mimics real-world data and is used for testing purposes.

- **Ensures privacy and data protection compliance:** GDPR, HIPAA, PCI-DSS and SOX compliant data generation
- **Speeds Up Data Provisioning:** Test environments get fresh data faster (no waiting for DB copies or manual anonymization
- **Usable by non-experts**. Realistic fake data can only be generated by experts, who know the precise rules governing the data set. However, anyone can generate synthetic data.
- **Increases adaptability.** Applications/data will inevitably change. It is easy to update synthetic data by simply retraining the ML model with newer data.
- **Cost reduction:** Decreased use of production data in test environments
- **Security & Privacy:** Eliminate risk of exposure of real data through synthetic anonymization
- **Abundant & On-Demand:** Quickly generate large volumes of data for load tests or extensive scenario coverage
- **Covers Edge Cases**: Simulate rare events or extreme values that might not yet exist in real data



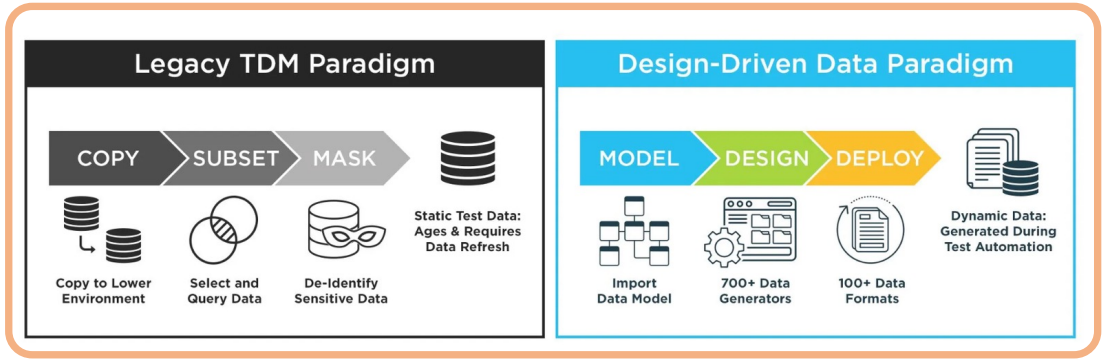
Limitations on the range of test data create limitations on coverage.

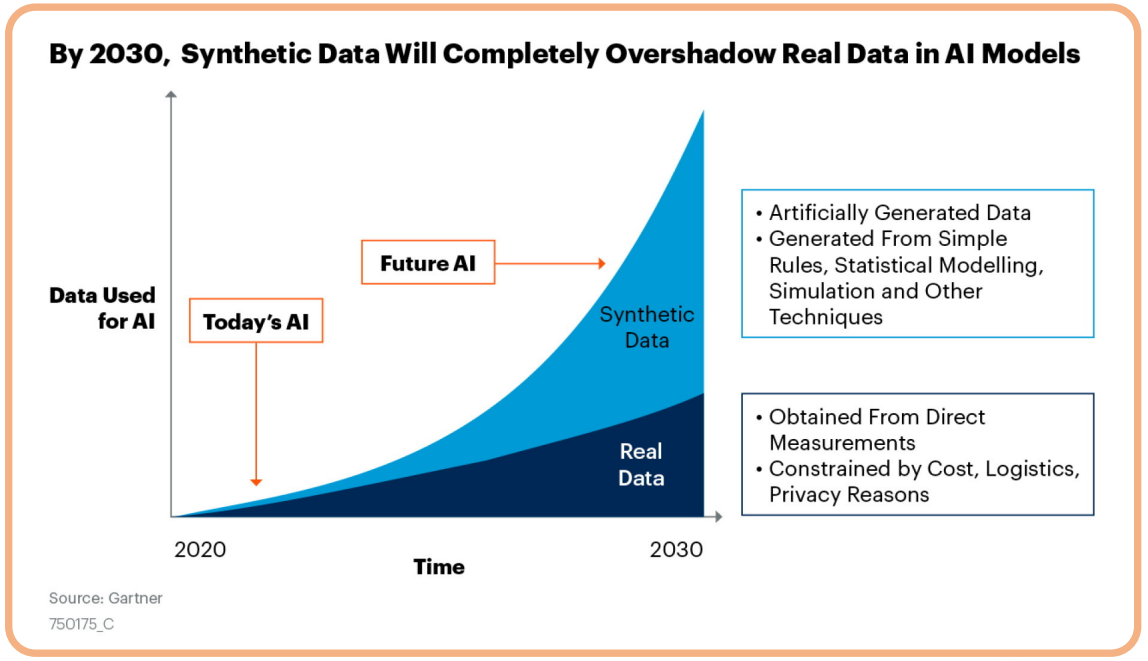
As a result, most organizations only test **20-30% of their code before deploying it to production.**

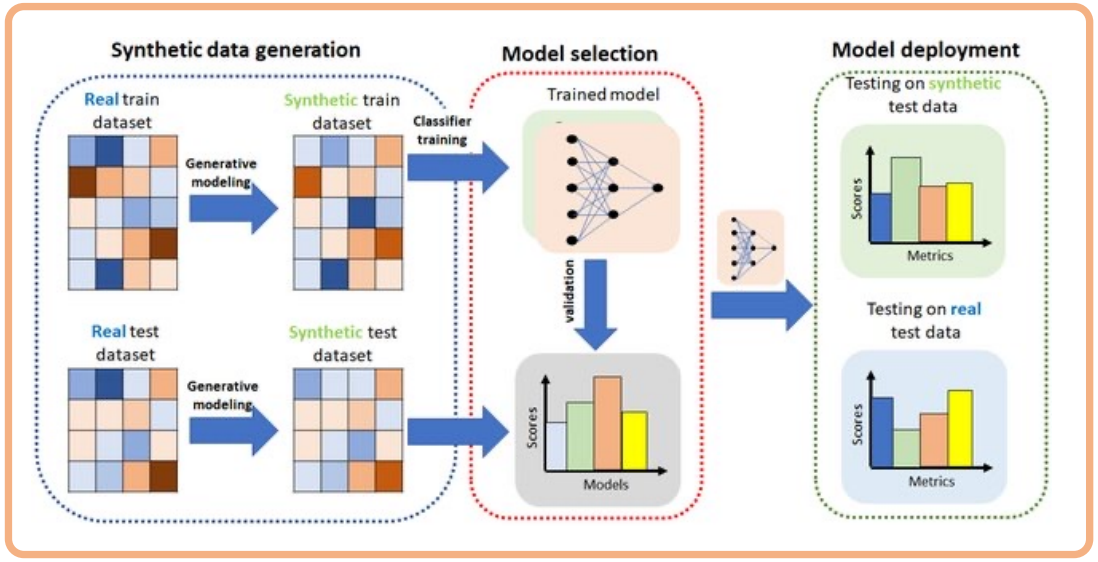


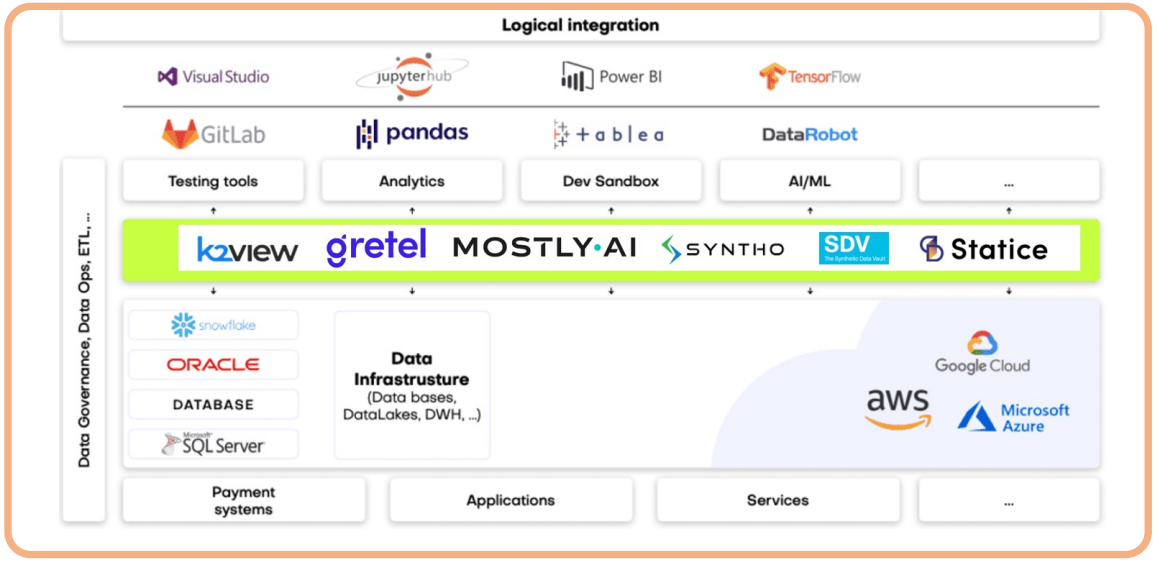
Synthetic Trends

AI, Data, and Chaos at the Helm of Software Quality









Demo – Generating Synthetic Data

AI, Data, and Chaos at the Helm of Software Quality

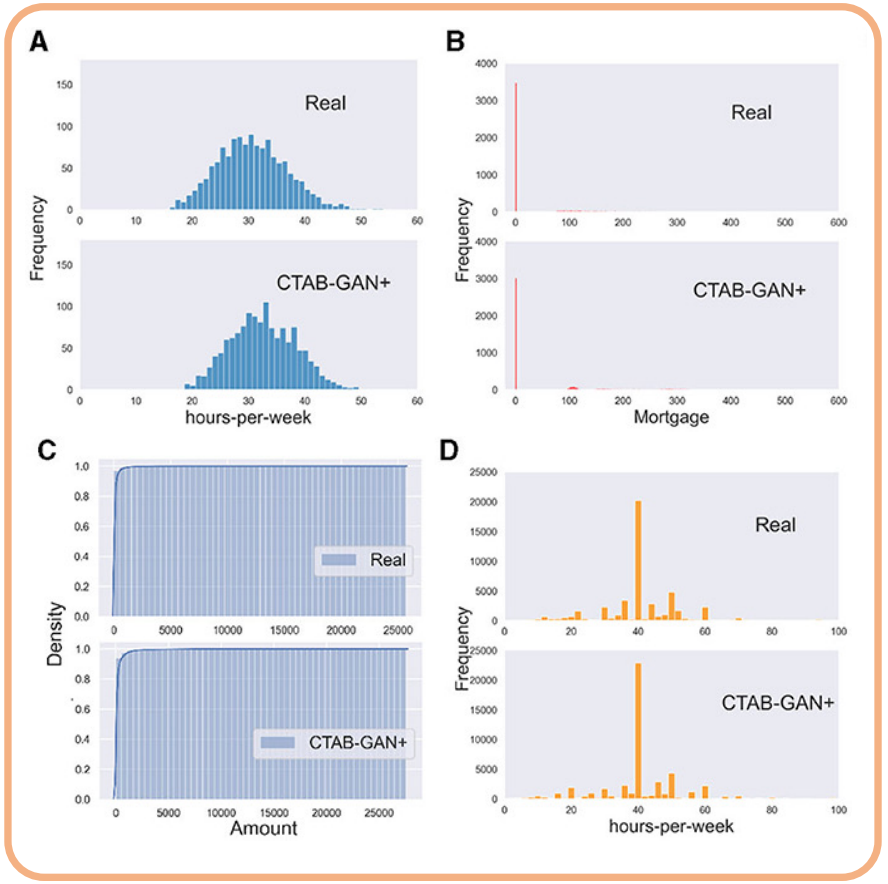
CTAB-GAN (Conditional Tabular GAN with Auxiliary Information and **Boosting)**

An enhancement of CTGAN. It introduces additional auxiliary information to significantly improve the fidelity and utility of the generated data. It incorporates improved conditional training procedures and advanced data encoding techniques.

Auxiliary Classifier & Conditional Information: Uses an auxiliary classifier to guide the generator, improving class-conditional data generation, especially effective for imbalanced classes.

Boosting Strategy: Implements a boosting strategy (akin to AdaBoost) to enhance the generator's performance iteratively, further refining the quality of synthetic data.

Improved feature encoding methods and training optimizations for enhanced data fidelity.



SDV
The Synthetic Data Vault

Better handling of highly **imbalanced** datasets.

Generates data with **higher accuracy and utility** (particularly relevant for downstream tasks such as **predictive** modeling).

AI-Powered Observability

Proactively monitor and detect subtle performance issues and regressions

AI-Powered Observability

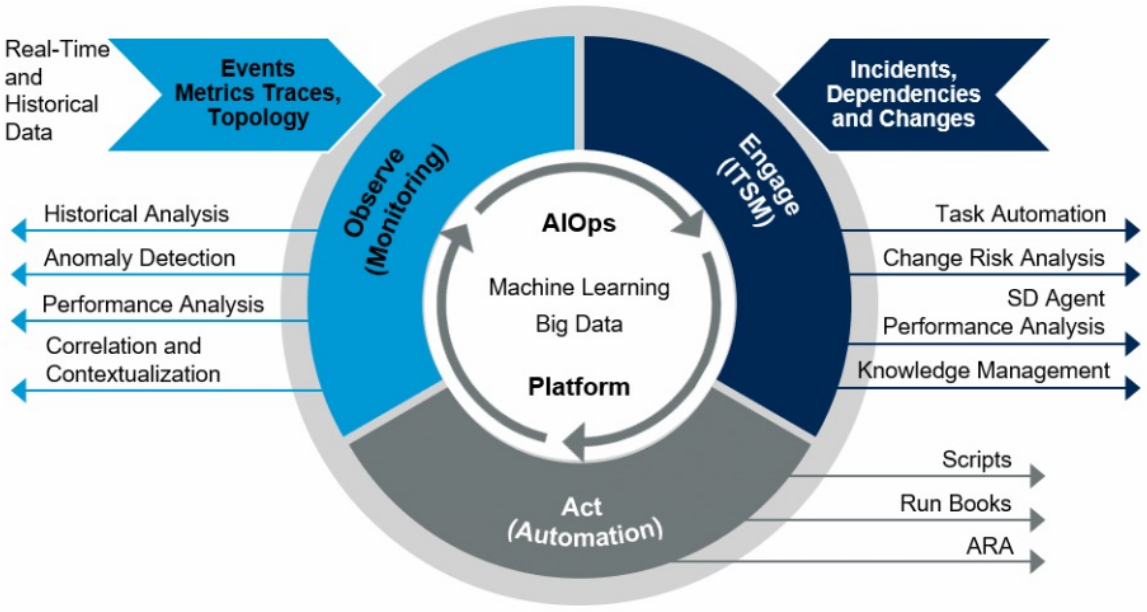
AI, Data, and Chaos at the Helm of Software Quality

What is Observability? Monitoring the health of software via logs, metrics, traces (think performance data, errors, user behavior)

- **Traditional Monitoring vs AI:** rule-based alerts and dashboards.
- **AI observability:** uses machine learning to detect anomalies, predict issues, and find root causes in vast data
- **QA Perspective:** Quality doesn't stop at release – need to ensure production behaves correctly. AI helps catch issues before customers notice
- **Feedback loop –** production insights inform better tests; proactive incident prevention is part of quality engineering

CHALLENGE

Consider this all-too-familiar challenge: An anomaly in a large microservice application triggers a storm of alerts as services around the globe are impacted. As your application contains literally millions of dependencies, how do you find the original error?



Gartner

Capabilities of AI in Observability

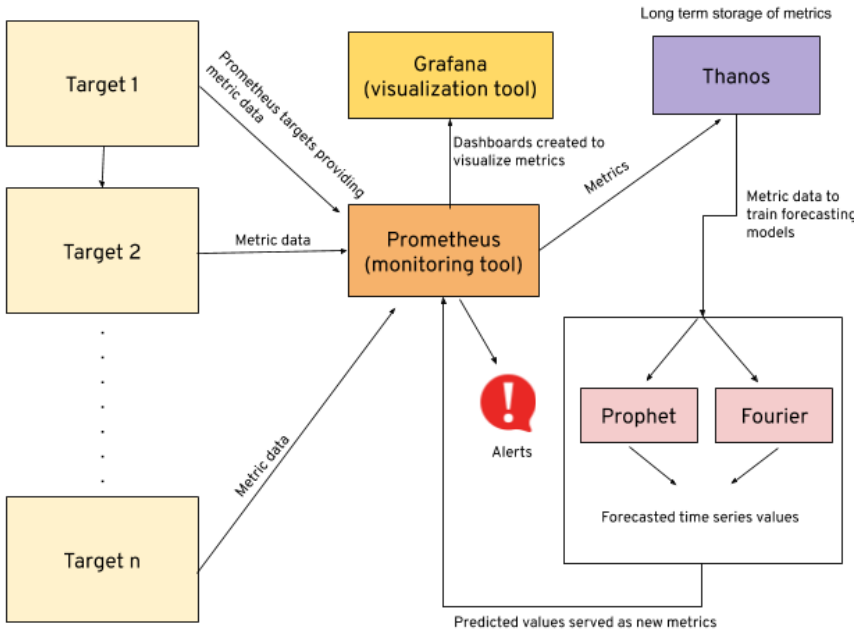
AI, Data, and Chaos at the Helm of Software Quality

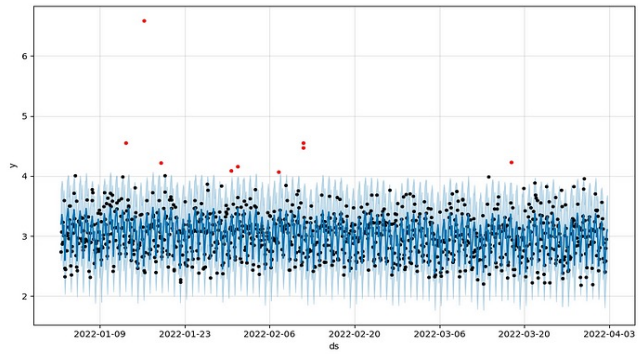
Anomaly Detection: ML models learn normal patterns (e.g., traffic, latency) and alert on deviations (e.g., sudden error spike)

Predictive Alerts: Forecast future trends (disk filling up, response time degradation) using models like Prophet – fix issues before they happen

Noise Reduction: Smart alerting reduces false alarms by correlating events (grouping related alerts, ignoring benign fluctuations)

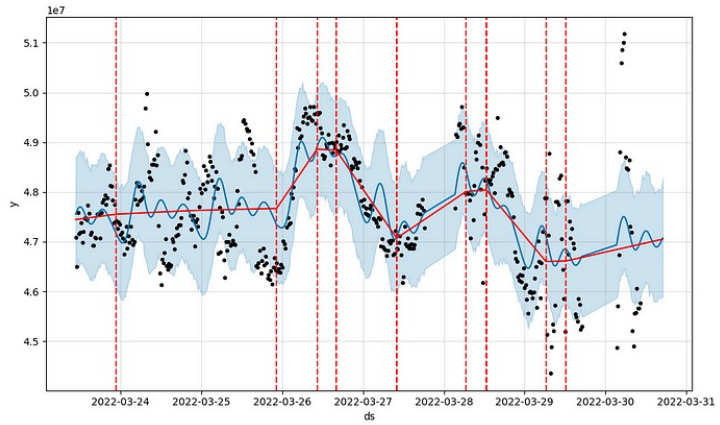
Root Cause Analysis: NLP + graph analysis on logs/traces to pinpoint likely cause of an incident (e.g., "Service X failure likely due to DB timeout")





Here is the **processed** data, where the anomalies are put as red dots

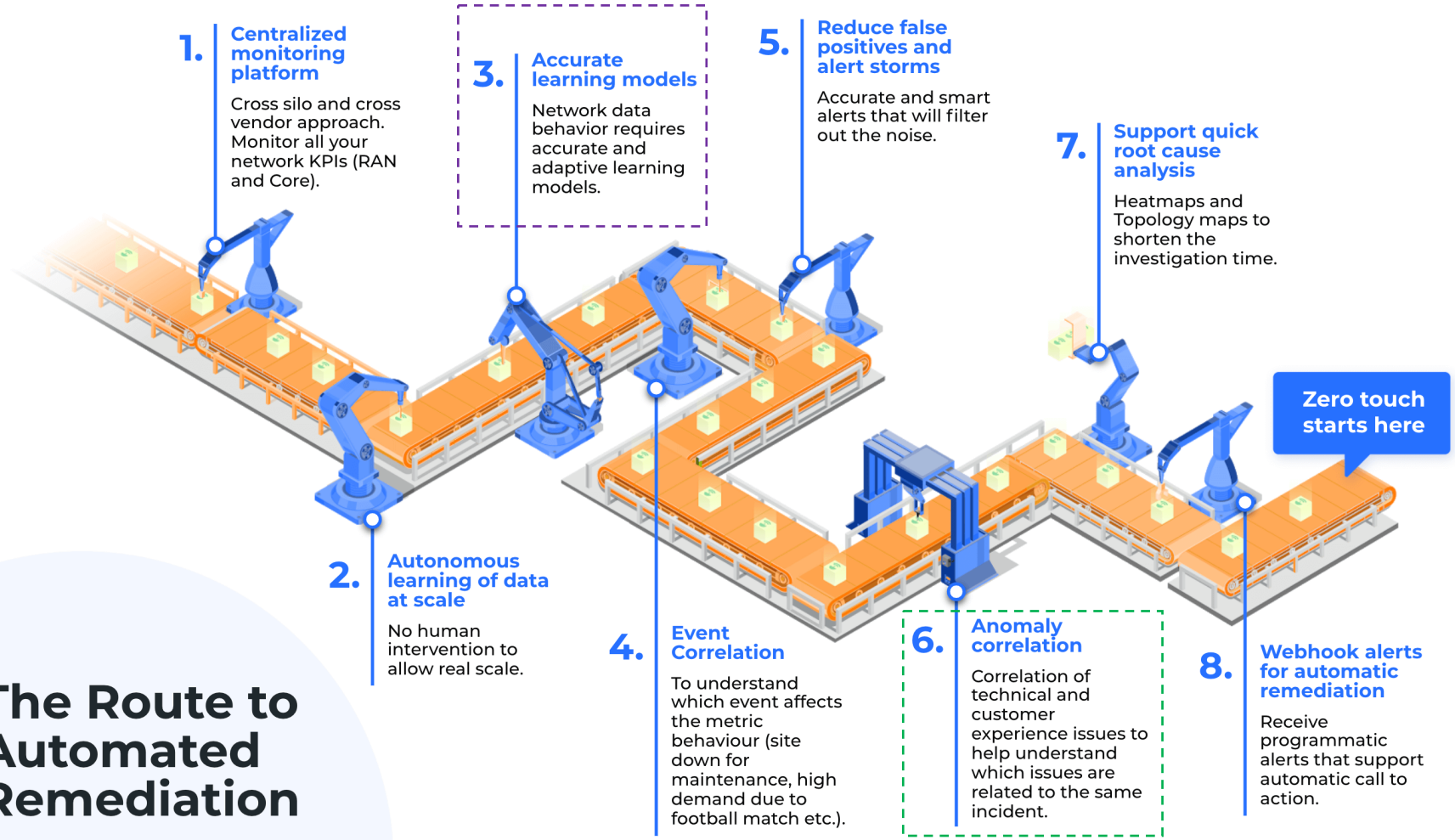
```
m =
Prophet(changepoint_prior_scale=0.05,changepoint_range=1,interval_width=.95)
```



Capabilities of AI in Observability

AI, Data, and Chaos at the Helm of Software Quality

Auto-Remediation (emerging): trend in AI/ML for observability that involves using intelligent automation tools to diagnose and fix issues automatically.



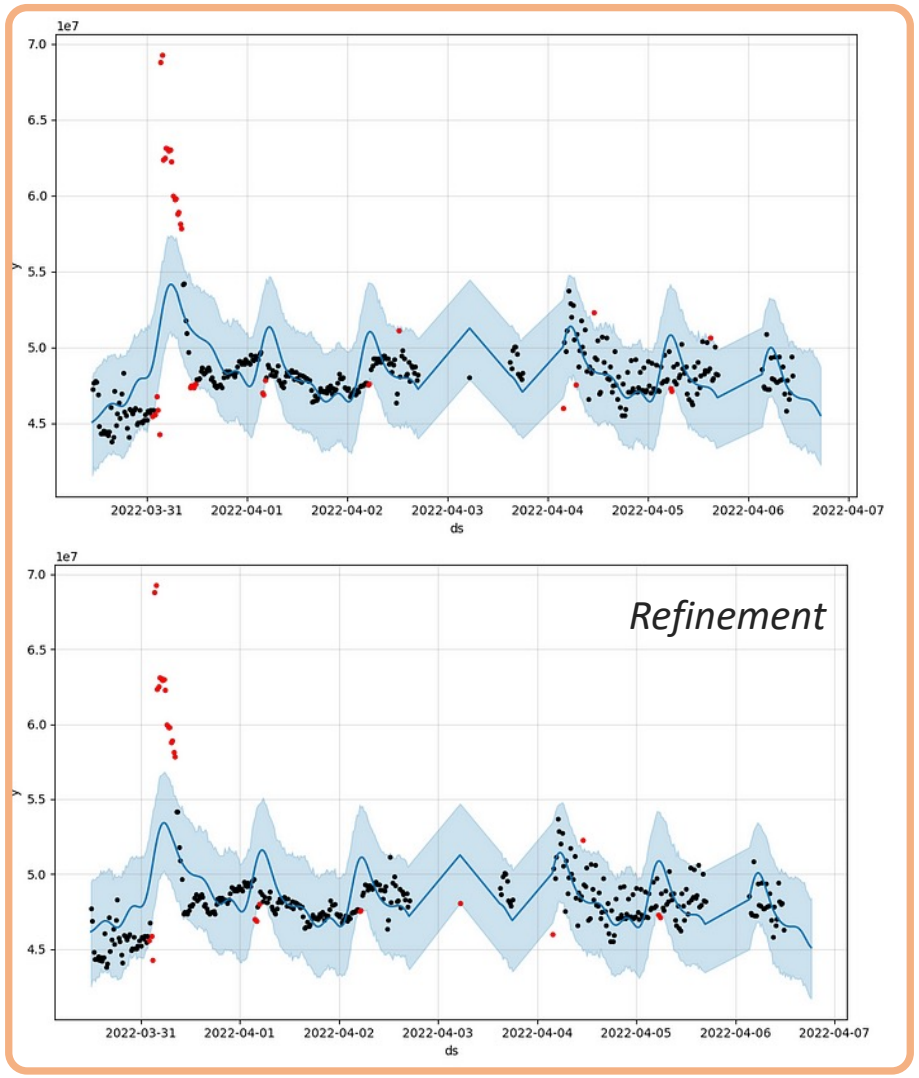
The Route to Automated Remediation

Anomaly Detection on Metrics

AI, Data, and Chaos at the Helm of Software Quality

Facebook (Meta) Prophet is good in that it requires minimal hyperparameter tuning; especially for system metrics data.

m = Prophet(changepoint_prior_scale=0.05,
changepoint_range=1, # By default changepoints are only inferred for the first 80% of the time series; In our case we are not forecasting # but fitting so we need to give 100 percent here
daily_seasonality=True,weekly_seasonality=False,yearly_seasonality=False, # we do not need to explicity set this; it takes from the data
seasonality_mode='additive', # this is the default - other option is 'multiplicative'
holidays_prior_scale=0.001 # default is 10, we set it to .001 to dampen holidays)



Anomaly Detection on Metrics

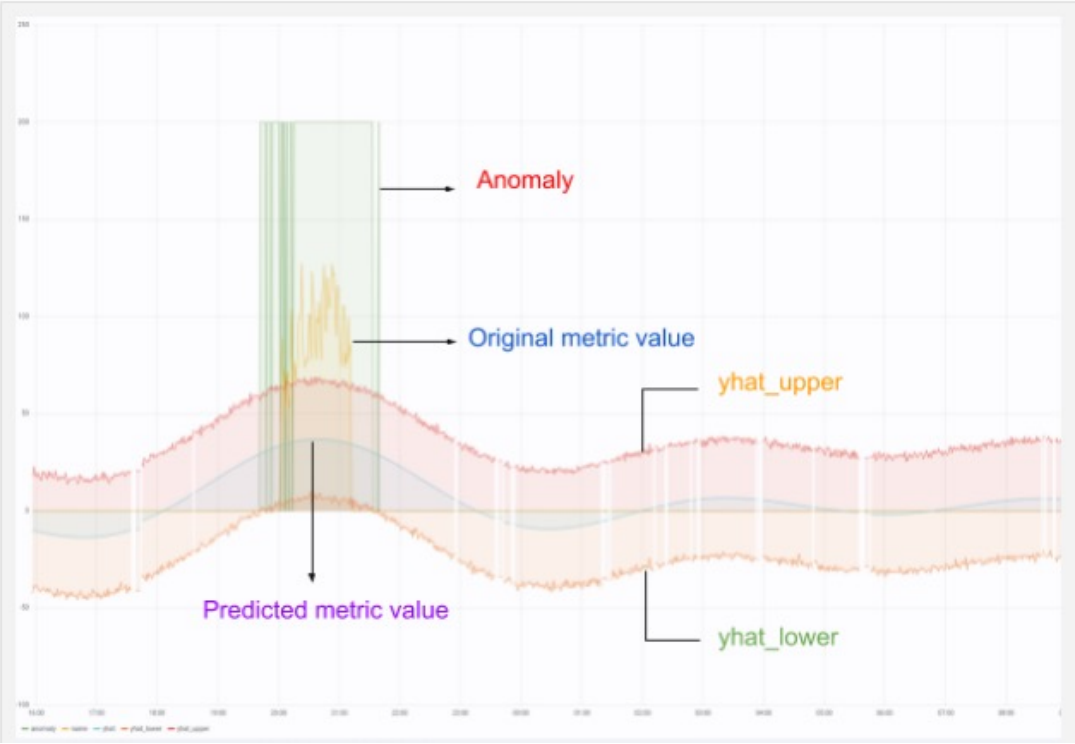
AI, Data, and Chaos at the Helm of Software Quality

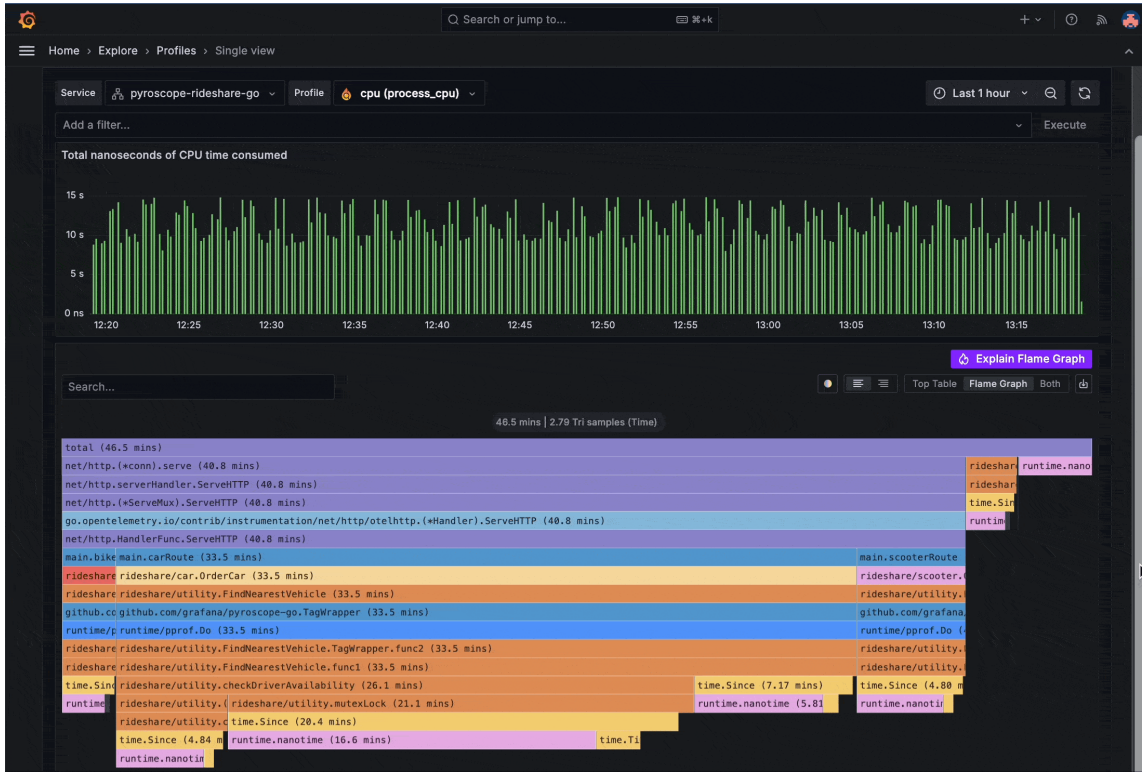
Explainable AI

Reduce repetitive manual tasks with machine learning/AI to minimize the toil of maintaining healthy services

Get faster incident response times with automated checks to help identify anomalies and automated workflows to correlate issues.

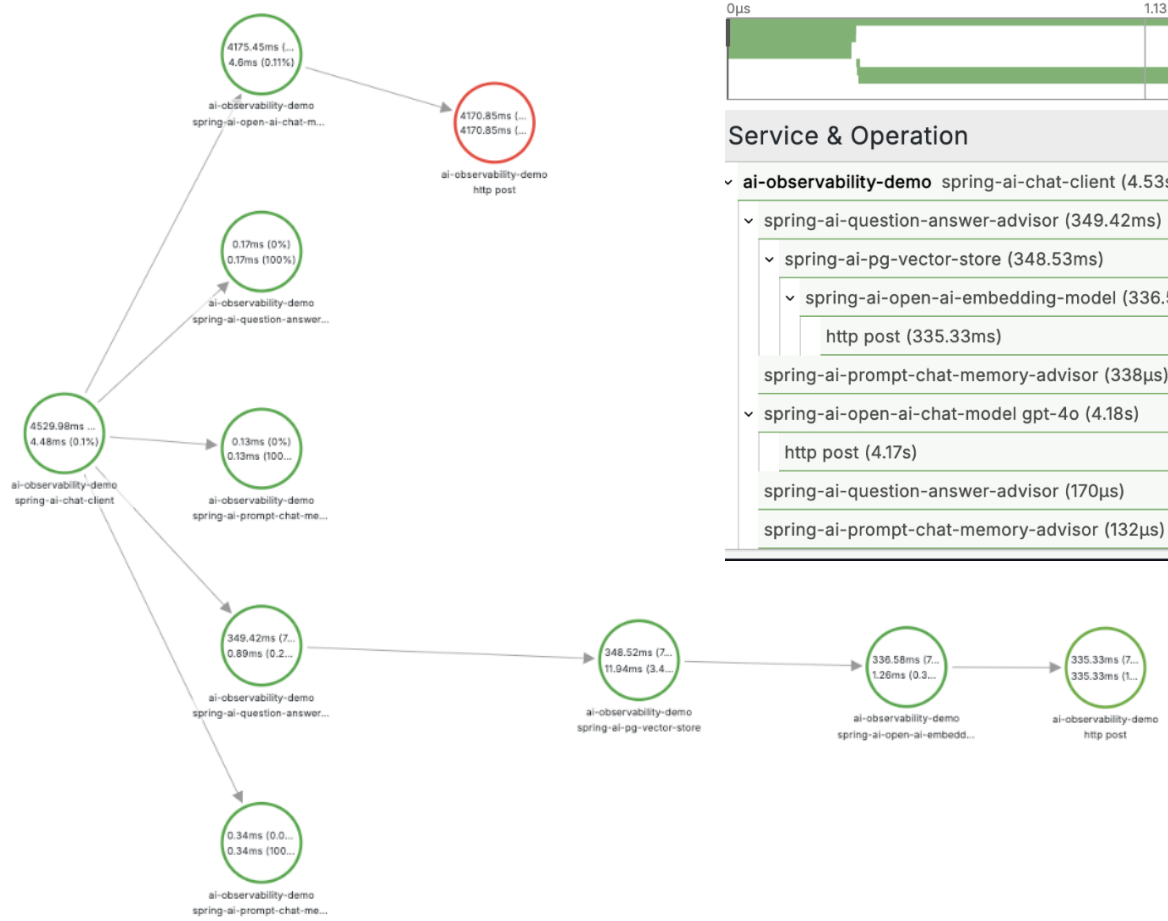
XAI refers to the development of AI systems that can provide clear and understandable explanations for their decisions and actions

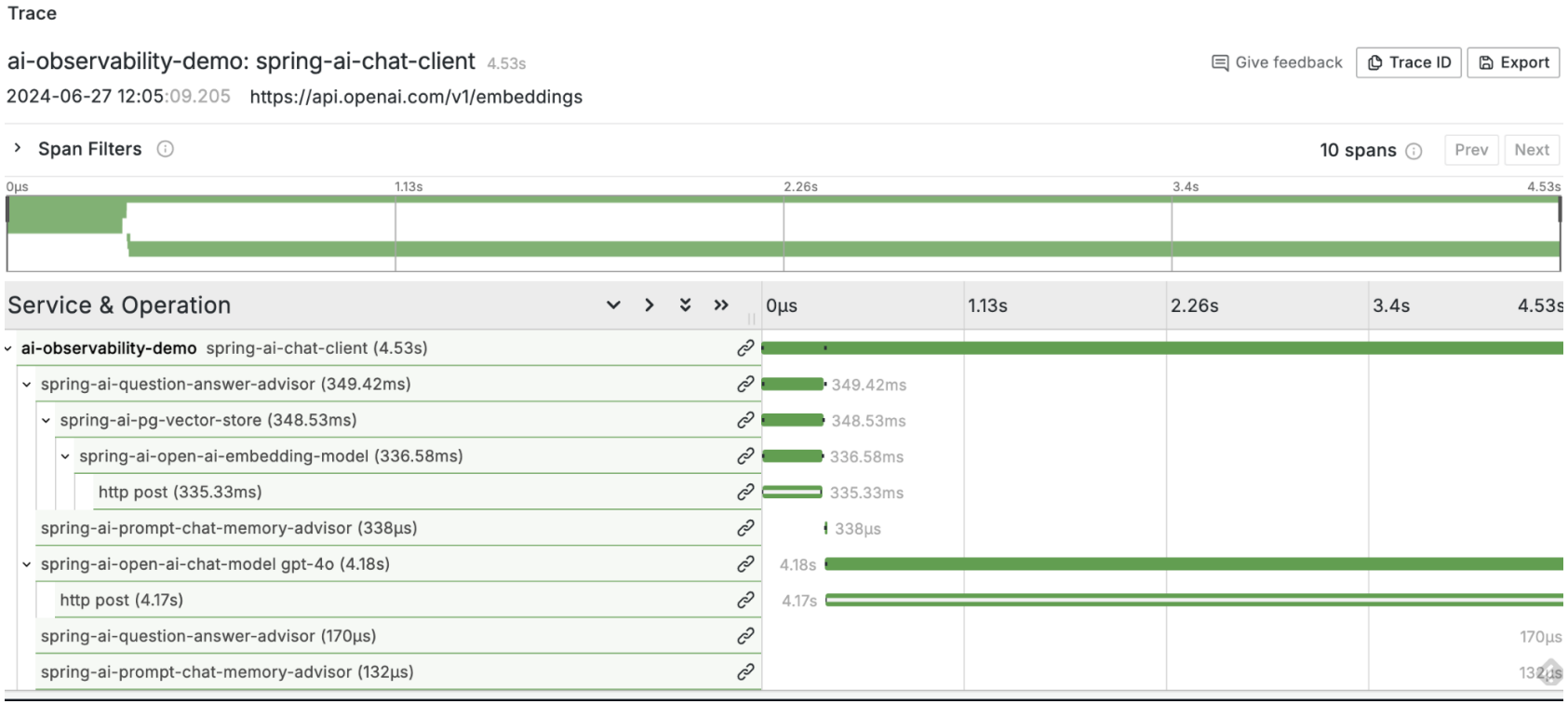


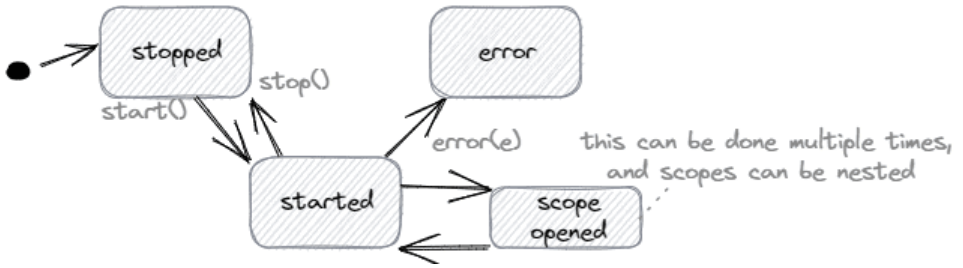


Anomaly Detection on Metrics

AI, Data, and Chaos at the Helm of Software Quality







Total time (% of trace) Self time (% of total) Self time / Trace duration

Intelligent Chaos Engineering

Dynamically generate failure scenarios
based on historical outages or user patterns

Intelligent Chaos Engineering

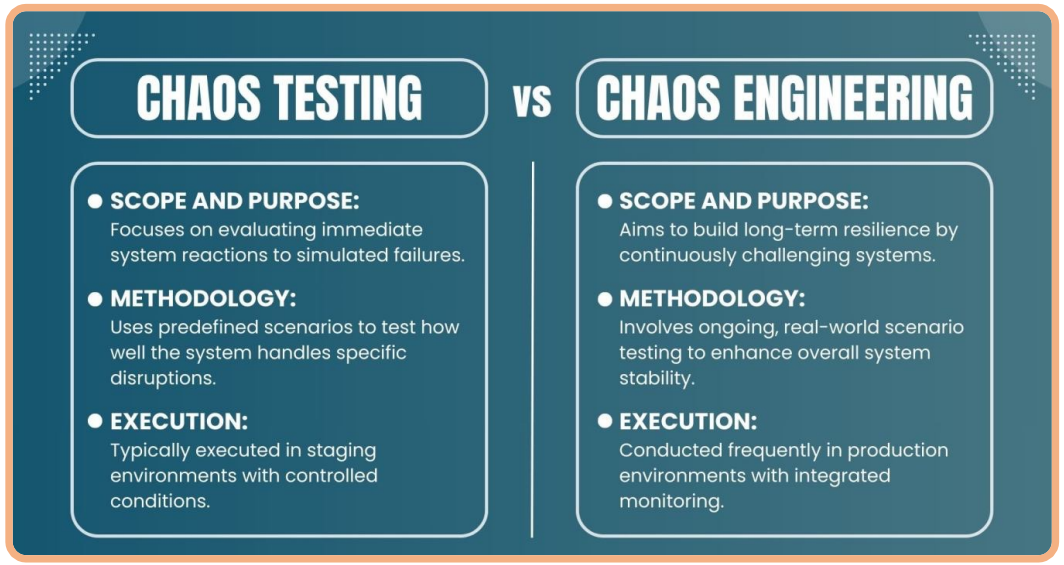
AI, Data, and Chaos at the Helm of Software Quality

Practice of deliberately causing failures (crashes, network issues, etc.) in a system to test its resilience

Why "Intelligent"? Traditional chaos tests are manually designed. With AI, we can analyze systems to smartly target weaknesses and automate experiments

Build confidence that software can withstand real-world incidents – uncover hidden fragilities under stress

Evolution: From simple chaos (randomly kill a server) to AI-driven chaos (choose faults based on architecture analysis, monitor impacts with ML)



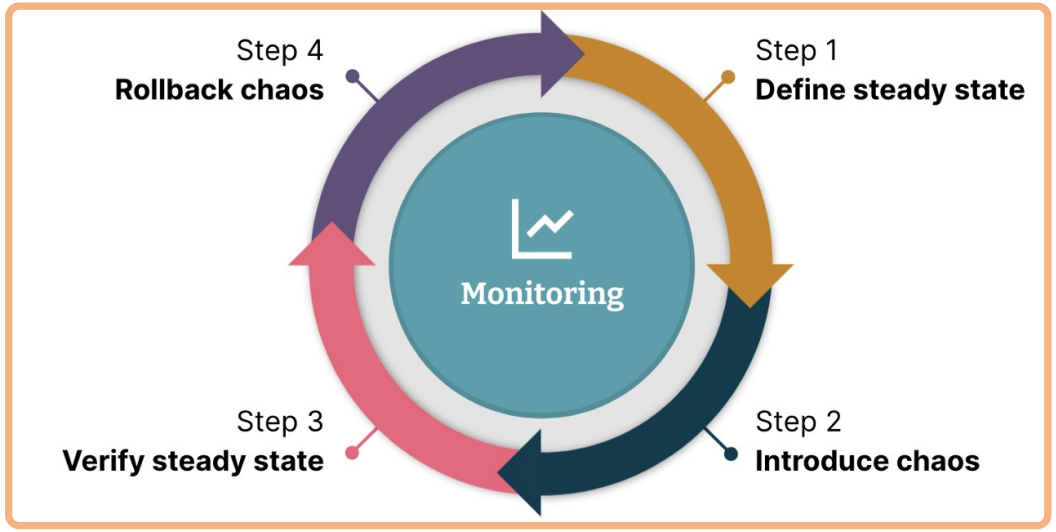
There are 4 major steps for running any chaos test.

1. The first step is defining a steady state, which means defining how an ideal system would look like. For a web application, the home page is returning a success response, for a web service this would mean that it is healthy, or it is returning a success for the health endpoint.

2. The second step is actually introducing chaos such as simulating a failure such as a network bottleneck / disk fill etc.

3. The third step is to verify a steady state, i.e, to check if the system is still working as expected.

4. The fourth step which is the most important step (more important if you are running in production) is that we roll back the chaos that we caused.



Chaos Engineering in Practice

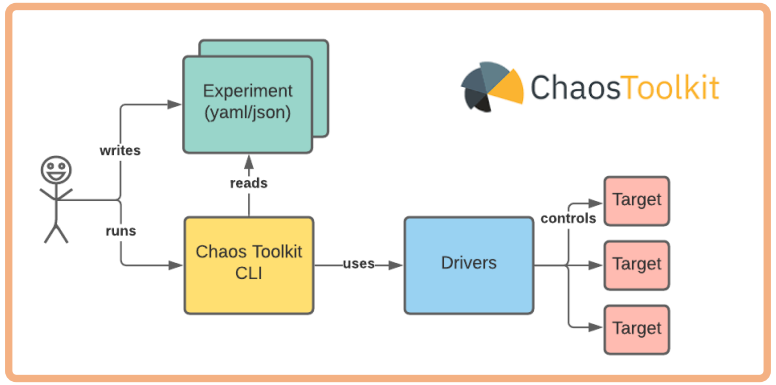
AI, Data, and Chaos at the Helm of Software Quality

Experiments = Steady State + Fault Injection: Define normal operation metrics (“steady state”), then introduce a fault (e.g., kill service, add latency) and observe

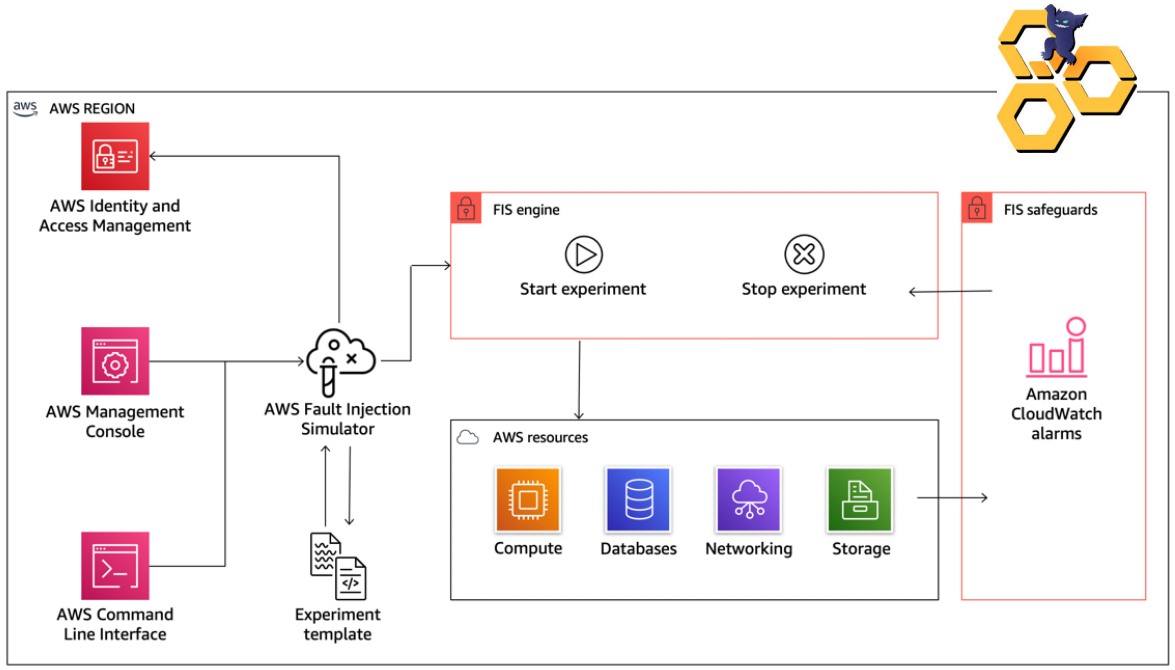
Often done in staging or with safeguards **in production** – ensure experiments don’t harm real users

Team Mindset: “If it can break, let’s break it ourselves first.” – encourages designing for failure, building auto-healing capabilities

Continuous Resilience Testing: Mature organizations run chaos tests regularly (even in CI/CD), not just once – ensures new changes don’t reintroduce fragility



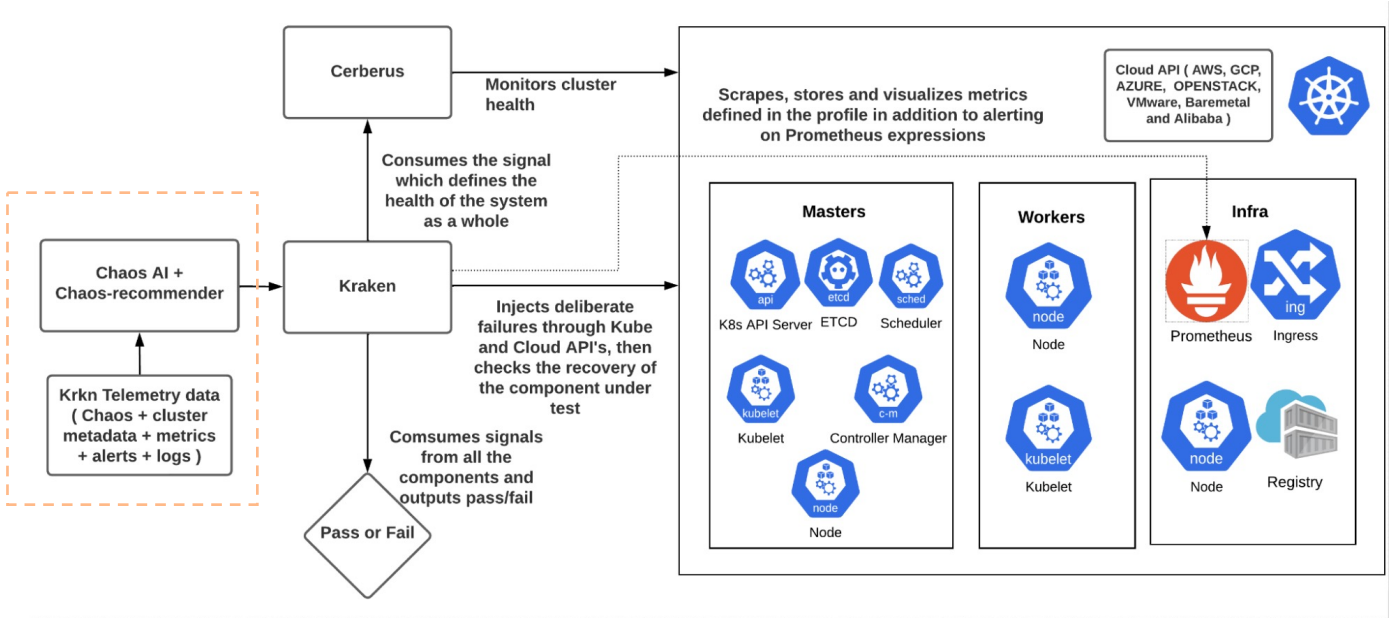
- ✓ **Chaos Monkey**: This tool randomly terminates virtual machine instances
- ✓ **Latency Monkey:** This tool simulates network latency
- ✓ **Chaos Kong:** This tool tests the resilience of Netflix’s data storage system by randomly killing entire data center regions
- ✓ **Janitor Monkey:** This tool identifies and deletes unused resources

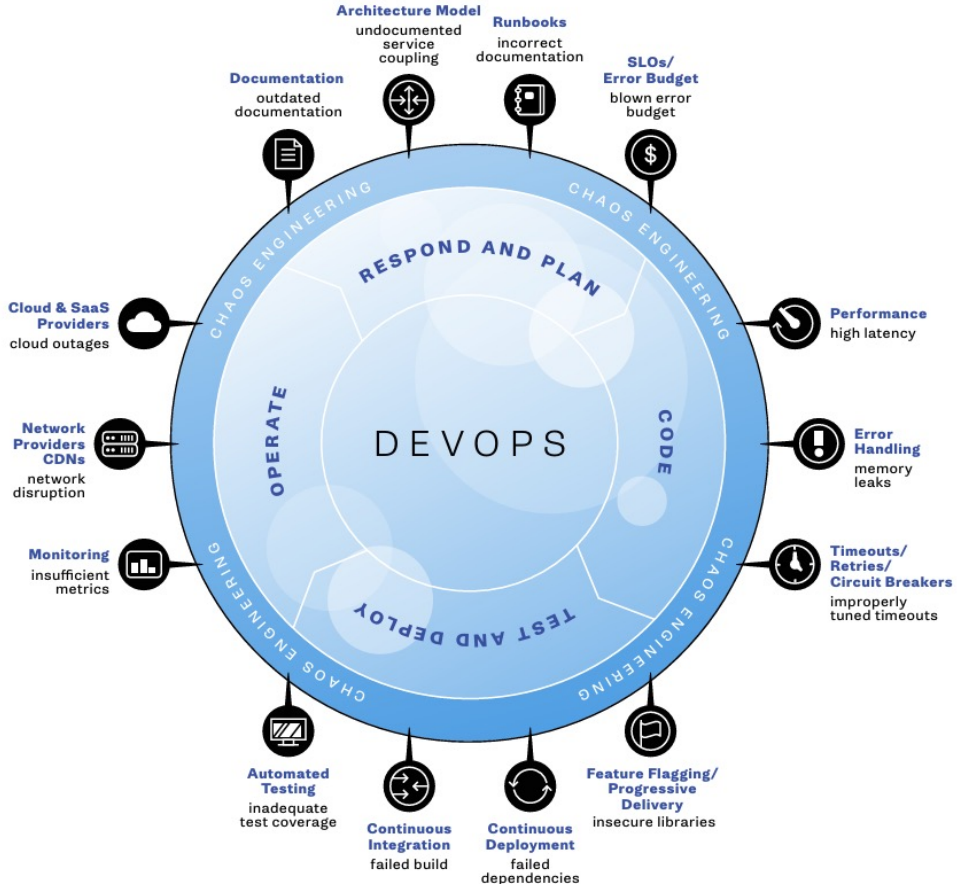


Chaos Engineering in Practice

AI, Data, and Chaos at the Helm of Software Quality

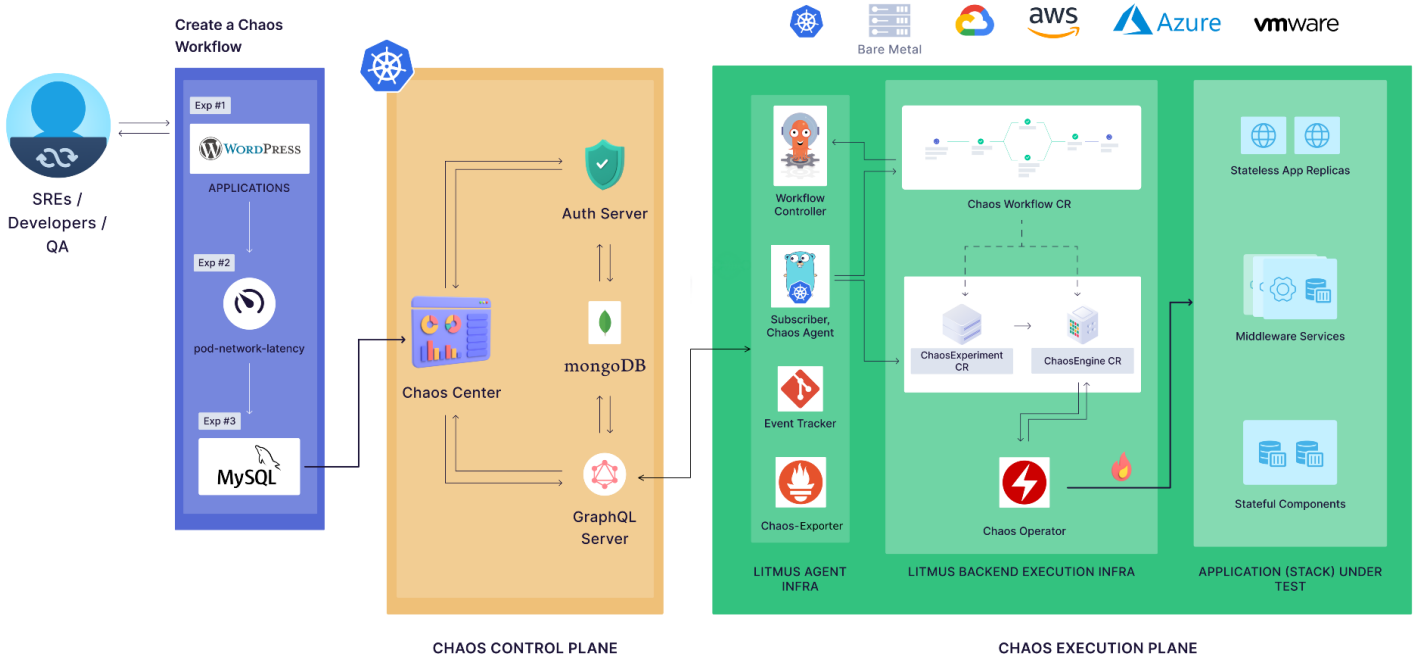
Litmus is an **open-source** Chaos Engineering platform that enables teams to identify weaknesses & potential outages in infrastructures by inducing chaos tests in a controlled way

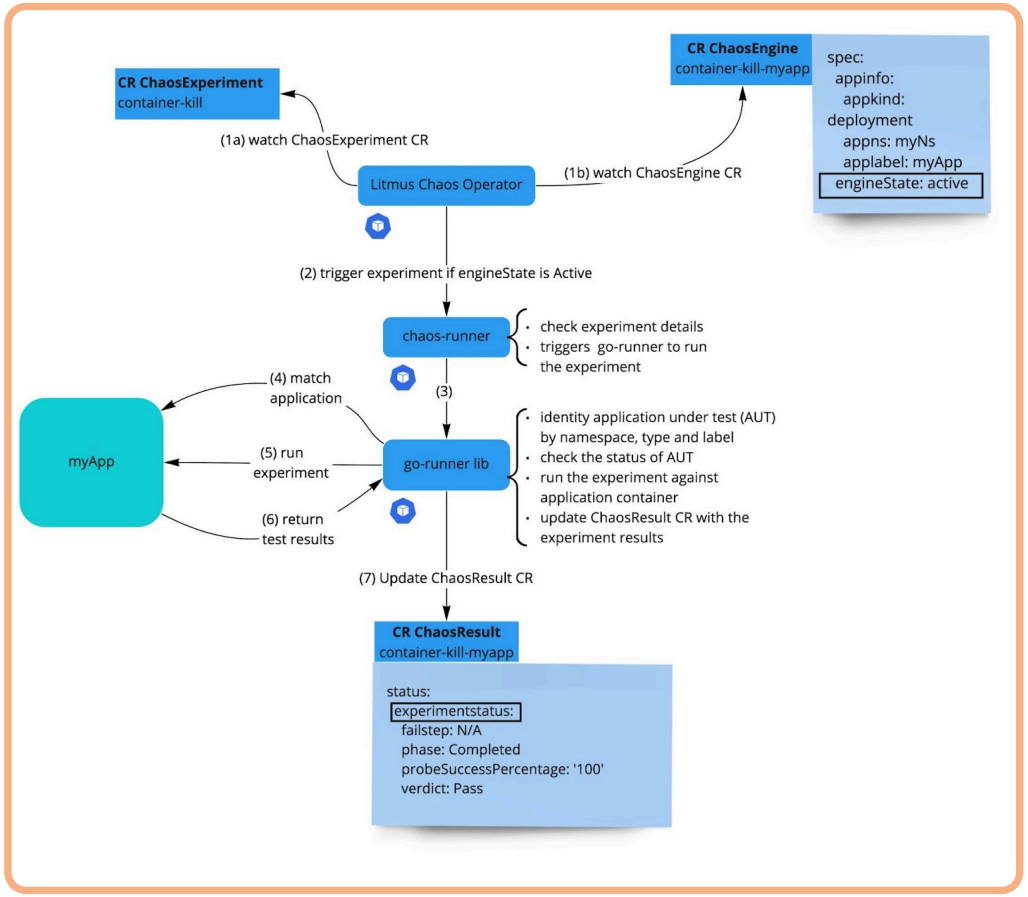




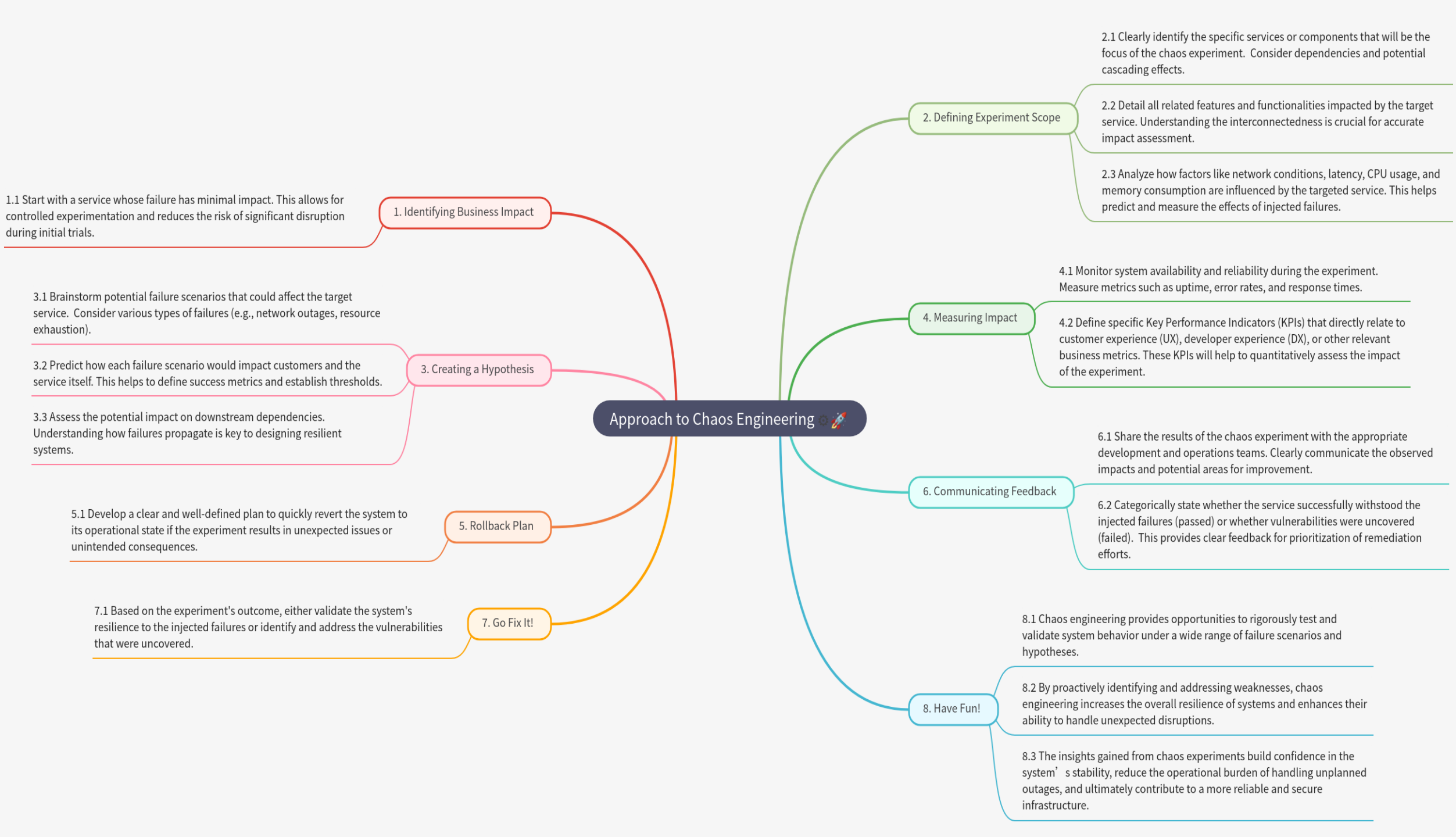
Chaos Engineering in Practice

AI, Data, and Chaos at the Helm of Software Quality





- **Real-time AI-powered data analysis** within Litmus Edge data flows, supporting major cloud services like OpenAI, Google Gemini, and AWS Bedrock, and on-premises integrations via OpenRouter, enabling real-time AI augmented data analysis, reasoning, and actions plans.
- **A structured API interface for Data and Metadata** that allows grounded AI agents to be integrated with Litmus Edge and Litmus UNS. Agents can now use historical data, context, and asset models from Litmus Edge for precise data interpretation with deeper context.
- **Optimized support for Nvidia GPU drivers**, enabling local runtime of Computer Vision and other GPU-intensive AI applications right at the edge.
- **Locally hosted SLMs (small language models)** now in beta, such as Microsoft Phi and Llama, to keep sensitive data secure in air-gapped environments – suitable for strict regulatory compliances.


Approach to Chaos Engineering 🚀
1. Identifying Business Impact
1.1 Start with a service whose failure has minimal impact. This allows for controlled experimentation and reduces the risk of significant disruption during initial trials.
2. Defining Experiment Scope
2.1 Clearly identify the specific services or components that will be the focus of the chaos experiment. Consider dependencies and potential cascading effects.
2.2 Detail all related features and functionalities impacted by the target service. Understanding the interconnectedness is crucial for accurate impact assessment.
2.3 Analyze how factors like network conditions, latency, CPU usage, and memory consumption are influenced by the targeted service. This helps predict and measure the effects of injected failures.
3. Creating a Hypothesis
3.1 Brainstorm potential failure scenarios that could affect the target service. Consider various types of failures (e.g., network outages, resource exhaustion).
3.2 Predict how each failure scenario would impact customers and the service itself. This helps to define success metrics and establish thresholds.
3.3 Assess the potential impact on downstream dependencies. Understanding how failures propagate is key to designing resilient systems.
4. Measuring Impact
4.1 Monitor system availability and reliability during the experiment. Measure metrics such as uptime, error rates, and response times.
4.2 Define specific Key Performance Indicators (KPIs) that directly relate to customer experience (UX), developer experience (DX), or other relevant business metrics. These KPIs will help to quantitatively assess the impact of the experiment.
5. Rollback Plan
5.1 Develop a clear and well-defined plan to quickly revert the system to its operational state if the experiment results in unexpected issues or unintended consequences.
6. Communicating Feedback
6.1 Share the results of the chaos experiment with the appropriate development and operations teams. Clearly communicate the observed impacts and potential areas for improvement.
6.2 Categorically state whether the service successfully withstood the injected failures (passed) or whether vulnerabilities were uncovered (failed). This provides clear feedback for prioritization of remediation efforts.
7. Go Fix It!
7.1 Based on the experiment's outcome, either validate the system's resilience to the injected failures or identify and address the vulnerabilities that were uncovered.
8. Have Fun!
8.1 Chaos engineering provides opportunities to rigorously test and validate system behavior under a wide range of failure scenarios and hypotheses.
8.2 By proactively identifying and addressing weaknesses, chaos engineering increases the overall resilience of systems and enhances their ability to handle unexpected disruptions.
8.3 The insights gained from chaos experiments build confidence in the system’s stability, reduce the operational burden of handling unplanned outages, and ultimately contribute to a more reliable and secure infrastructure.

Key takeaways

Quality at Speed: Embracing AI and automation (GenAI, synthetic data) enables faster testing and broader coverage

- QA keeps up with rapid delivery without sacrificing depth.

Proactive QA: Shift from reactive bug-finding to proactive prevention – monitoring and chaos testing catch issues before users do.

- Quality Engineering is a continuous lifecycle.

AI as QA Partner: AI isn't replacing testers – it's augmenting them. Testers become orchestrators of smart tools, focusing on strategy while AI handles grunt work and complex analysis.

Business Impact: These innovations lead to fewer production incidents, faster releases, and higher customer satisfaction.

- QA becomes a competitive advantage, not a bottleneck.

Future Vision: QA roles will evolve to include **data science and SRE skills**. The QA engineer of tomorrow designs AI models for testing, tunes observability alerts, and plans chaos experiments – truly at the helm of software quality in an AI-driven world.

Diogo Gonçalves Candeias

diogoguerramaio

dioguitomaio@gmail.com

diogo.goncalves@exceltic.com

GRACIAS


expoQA 24
MADRID
May 28th, 29th, 30th 2024
Thank you for attending
expoqa.com