A.3 Architectural Decision Records

This section contains the ADRs of the MiSArch Experiment Tool.

ADR 1: Abstraction of the Experiment Definition

Status: Accepted
Date: 10.04.2025

Context

An experiment consists of various definitions and configurations. These can be stored in their native formats (e.g., JSON, YAML, etc.) or through a unified Domain Specific Language (DSL). Native formats have the benefit not to introduce any limitations. A unified DSL makes experiment setups simpler and more understandable.

Decision

We choose to store experiments in their native formats to preserve complexity and flexibility for expert users.

Consequences

- Experts can define complex experiments without being limited by a DSL.
- Potentially higher learning curve for users not familiar with native formats.
- No information is lost due to abstraction.

ADR 2: Experiment Executor Component: Container vs Kubernetes Operator

Status: Accepted
Date: 10.04.2025

Context

The central *Experiment Executor* can be implemented either as a Kubernetes operator or as a standalone container. A standalone components is more flexible in different environments, whereas a native Kubernetes operator has more features and access when applied in a Kubernetes environment.

Decision

We implement the Experiment Executor as a container to ensure compatibility with Docker and managed cloud environments beyond Kubernetes.

Consequences

- Maintains compatibility across various deployment environments.
- Sacrifices some Kubernetes-native API integrations.

ADR 3: Dedicated Component for Gatling Execution

Status: Accepted
Date: 10.04.2025

Context

Gatling load testing could be integrated directly into the Experiment Executor or delegated to a separate component. A direct integration makes the setup of the project simpler, yet in terms of separation of concerns and possibly dedicated required resources for load generation the separation can be useful.

Decision

We use a dedicated component for Gatling execution to ensure resource separation and scalable load generation.

Consequences

- Clear separation of load generation responsibilities.
- Increased complexity due to more components.

ADR 4: Dedicated Component for the Chaos Toolkit Execution

Status: Accepted
Date: 10.04.2025

Context

The Chaos Toolkit uses a Python stack, whereas the Experiment Executor uses a Kotlin and Spring Boot stack. The integration could either be unified in a combined component or kept separate. Benefits of a combined component is simplicity during the experiment setup. However, it increases complexity in the one component and causes the loss of separation of concerns.

Decision

A dedicated component for the Chaos Toolkit will be used.

Consequences

- Complexity increases due to an additional component.
- Easier to bundle Python dependencies and extensions.
- Better separation of technology stacks.

ADR 5: Integration of Experiment Config Component

Status: Accepted
Date: 10.04.2025

Context

The existing MiSArch Experiment Config component could be integrated directly into the Experiment Executor or kept as a standalone service. Including it into the Experiment Executor would reduce the complexity of the architecture, keeping it as is keeps a better separation of concerns and lets us focus on creating value instead of refactoring.

Decision

We keep the Experiment Config component separate and communicate with it via its API.

Consequences

- Preserves separation of concerns.
- Adds one extra hop in execution flow.
- Reduces coupling between configuration and execution logic.

ADR 6: Storage and Provisioning of Gatling Metrics

Status: Accepted
Date: 10.04.2025

Context

Gatling produces HTML/JavaScript dashboards by default, which are not machine-readable. Those could be either directly visualized by integrating the HTML page into the MiSArch Experiment Frontend or parsed into a machine-readable format and injected into a time-series database such as InfluxDB. The former one is simpler and ensures visualization as designed by the Gatling developers, the latter one is more sustainable and allows for historic comparsion and fine-tuning of the metrics and dashboards.

Decision

Parse the generated HTML/JS output into a machine-readable format and inject it into InfluxDB.

Consequences

- Enables time-series storage and historic comparison.
- Increases processing complexity after test execution.
- Allows integration with Grafana and similar tools.

ADR 7: OpenTelemetry Instrumentation on Application Level

Status: Accepted
Date: 10.04.2025

Context

Instrumentation must be consistent across languages and frameworks, using the OpenTelemetry standard. This can be achieved by using different libraries or frameworks for each technology stack, or by auto-instrumentation using a Kubernetes operator, which however is limited to a Kubernetes infrastructure. They all do incorporate the latest version of the OpenTelemetry semantic conventions.

Decision

- Use latest OpenTelemetry Semantic Conventions for HTTP metrics.
- Use Java Agent for Kotlin + Spring Boot for ease of use.
- Use OpenTelemetry SDK (with experimental flag) for Node.js.
- Use community-maintained library for Rust + Axum with contributions to bring it up to standard.

Consequences

- Ensures unified metrics format across services.
- Introduces additional instrumentation and setup complexity.

ADR 8: Frontend Framework for Experiment Executor

Status: Accepted
Date: 10.04.2025

Context

The MiSArch project uses Vue.js, while the existing Experiment Config frontend is in Angular. Both are widely used frotend frameworks with slightly different approaches and complexity.

Decision

We adopt Vue.js for the Experiment Executor frontend for consistency with the MiSArch frontend, as the MiSArch Experiment Executor Frontend makes the MiSArch Experiment Config Frontend mostly obsolete..

Consequences

- Frontend stack remains consistent across applications.
- Reduces context-switching for frontend developers.