Bifrost Toolkit: Data-Driven Release Strategies

Formalize and automate real-time, data-driven live-testing methods using a DSL

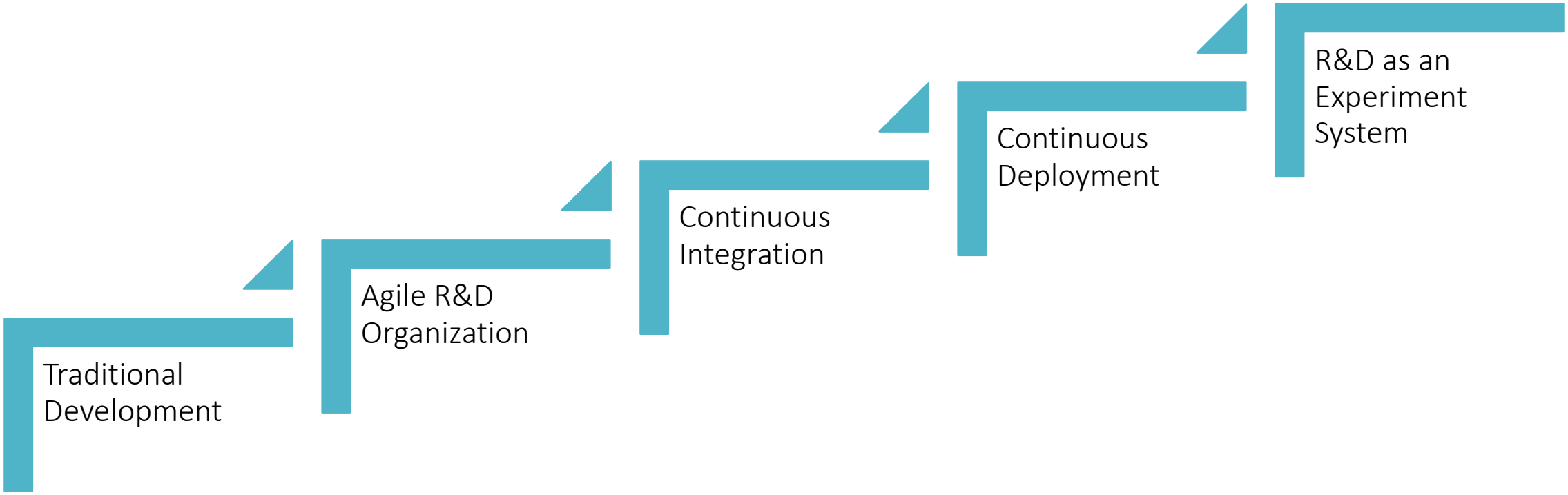
Master Thesis Defense – 28.04.2016

Dominik Schöni

s.e.a.l. software evolution & architecture lab

University of Zurich[UZH]

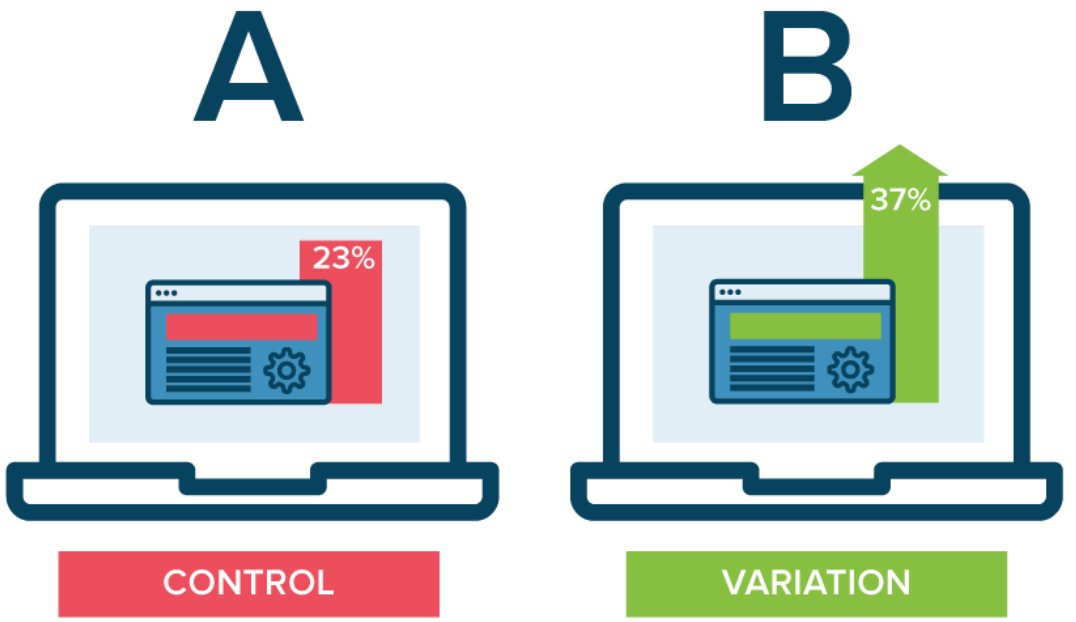
The Stairway to Heaven



[OAB12]

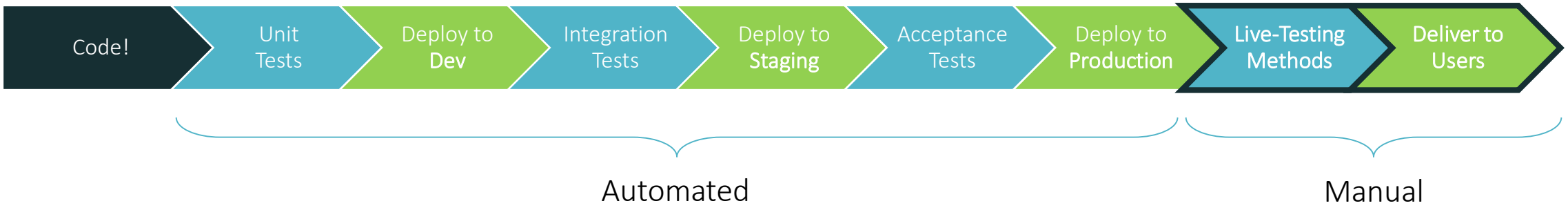
Live-Testing Methods

A/B Testing



Source: http://www.optimizely.com

Releasing Software



Automation is key

Automation allows for **transparency and traceability**

Gives developers **confidence when releasing**

Time to market gets **accelerated**

Computers are better at repeatable tasks than humans

Research Question #1

«How can we formalize a (generic) model for data-driven release and deployment strategies?»

Methods of Live-Testing

- A/B Testing:

  A way to compare two versions of software with each other, often only differentiated in one aspect

- Canary Launches:

  Introducing a new version into a stable environment

- Shadow Launches:

  Deploying features that are not visible to users

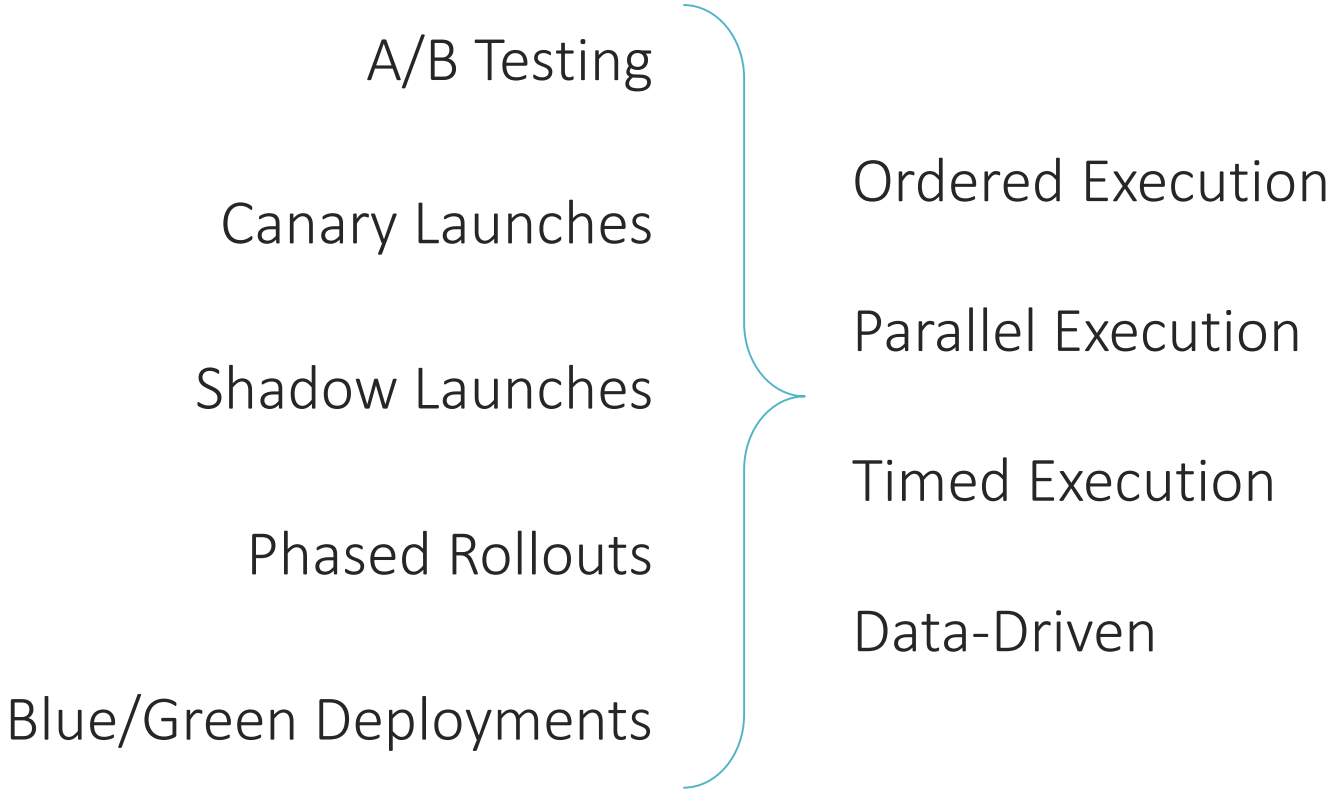
- Phased Rollouts:

  Expose users gradually to a new software version

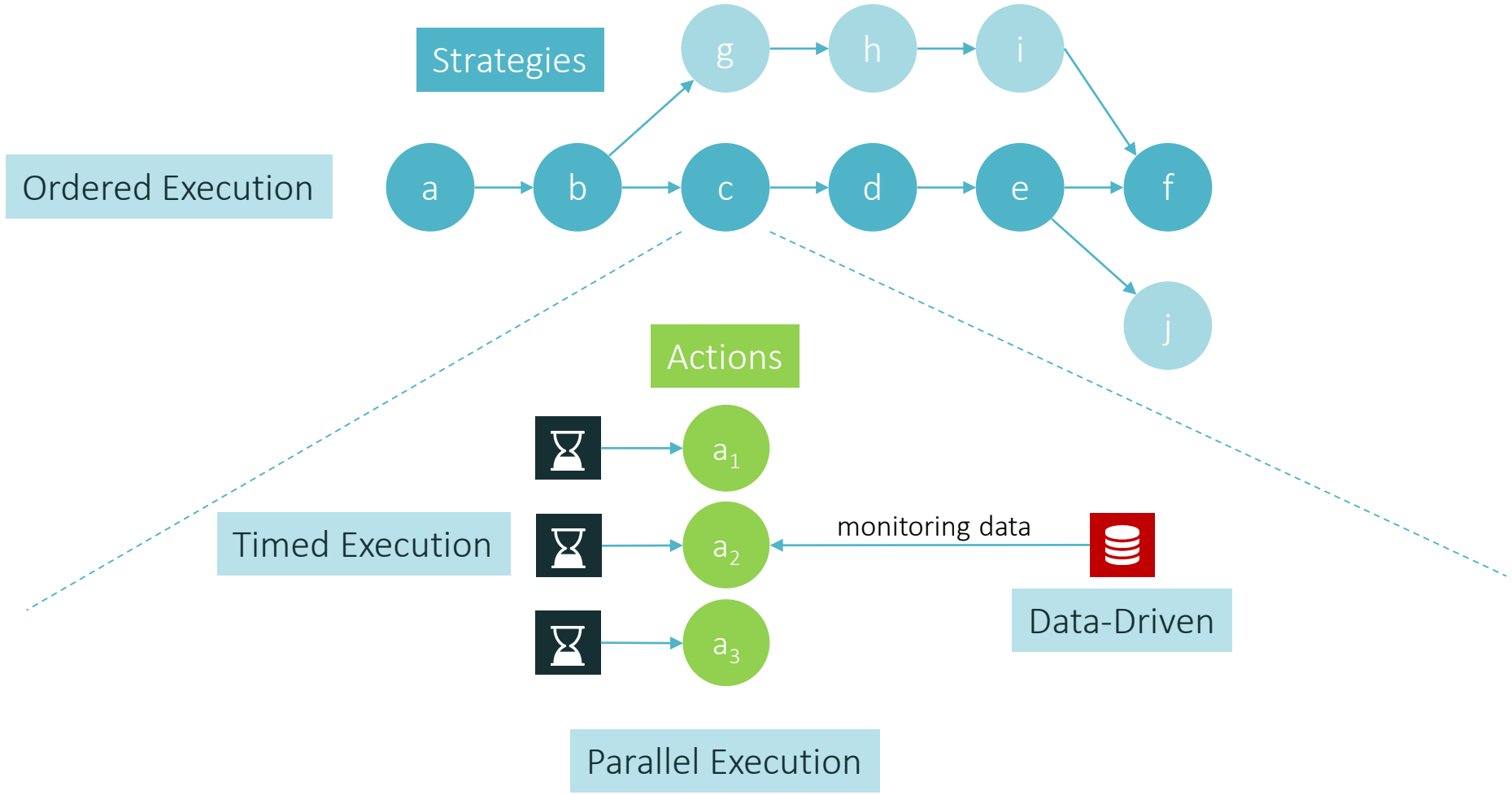
- Blue/Green Deployments:

  Having two identical production environments, allowing to switch seamlessly from one to another

Identifying characteristics



Bifrost Release Model



Bifrost Release Model

The model was formalized using a mathematical representation:

$$Release: \quad \{\{c_1, \ldots, c_n\}, (s_1, \ldots, s_n)\}$$

$$Strategy: \quad \{Parallel_A, Success_A, Failure_A\}$$

$$Action: \quad A = \{\Theta, \Delta, \Omega\}$$

$$Parallel_A: \quad \{a_1, \ldots, a_n \mid a_i \in Action\}$$

$$Success_A: \quad \{x \mid x \in Action\}$$

$$Failure_A: \quad \{x \mid x \in Action\}$$

....

Research Question #2

«How can we build a tool that supports and automates data-driven release and deployment strategies for microservices-based architectures in a non-intrusive way?»

Leveraging the Bifrost Release Model

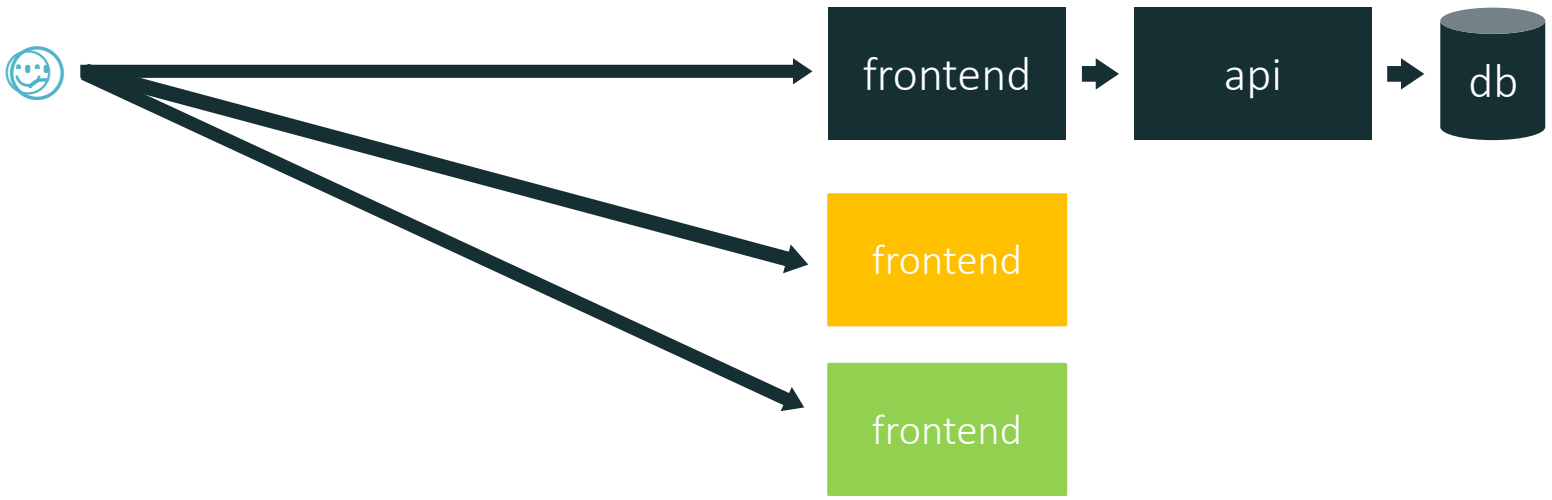
The model identifies abstract concepts of **releases**, **strategies** and **actions.**

→ Translated for developers into a Domain Specific Language

Bifrost DSL: YAML-based DSL

- Specifies what services to modify during a release, and how.
- Includes concepts of **releases**, **strategies** and **actions** as in the Bifrost Release Model.
- Enables transparency and traceability as it is file-based and can be version-controlled.

Approach: Implementation Techniques

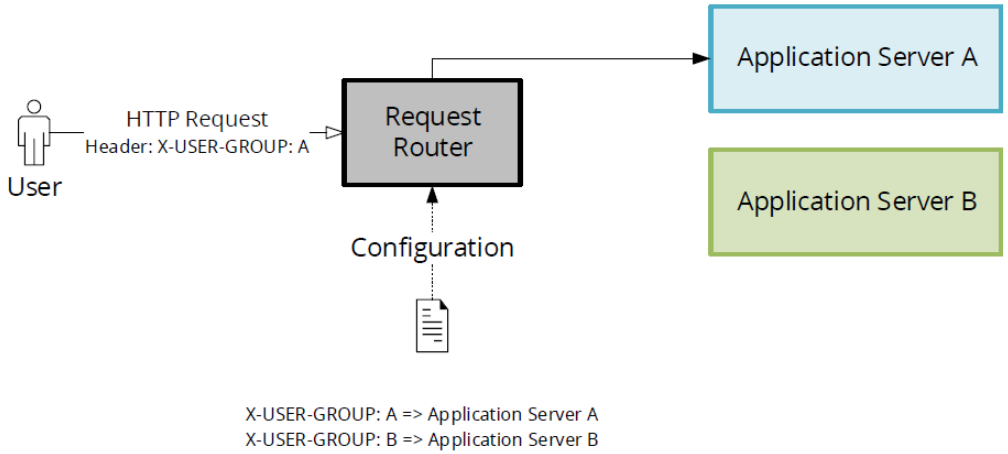


Approach: Implementation Techniques

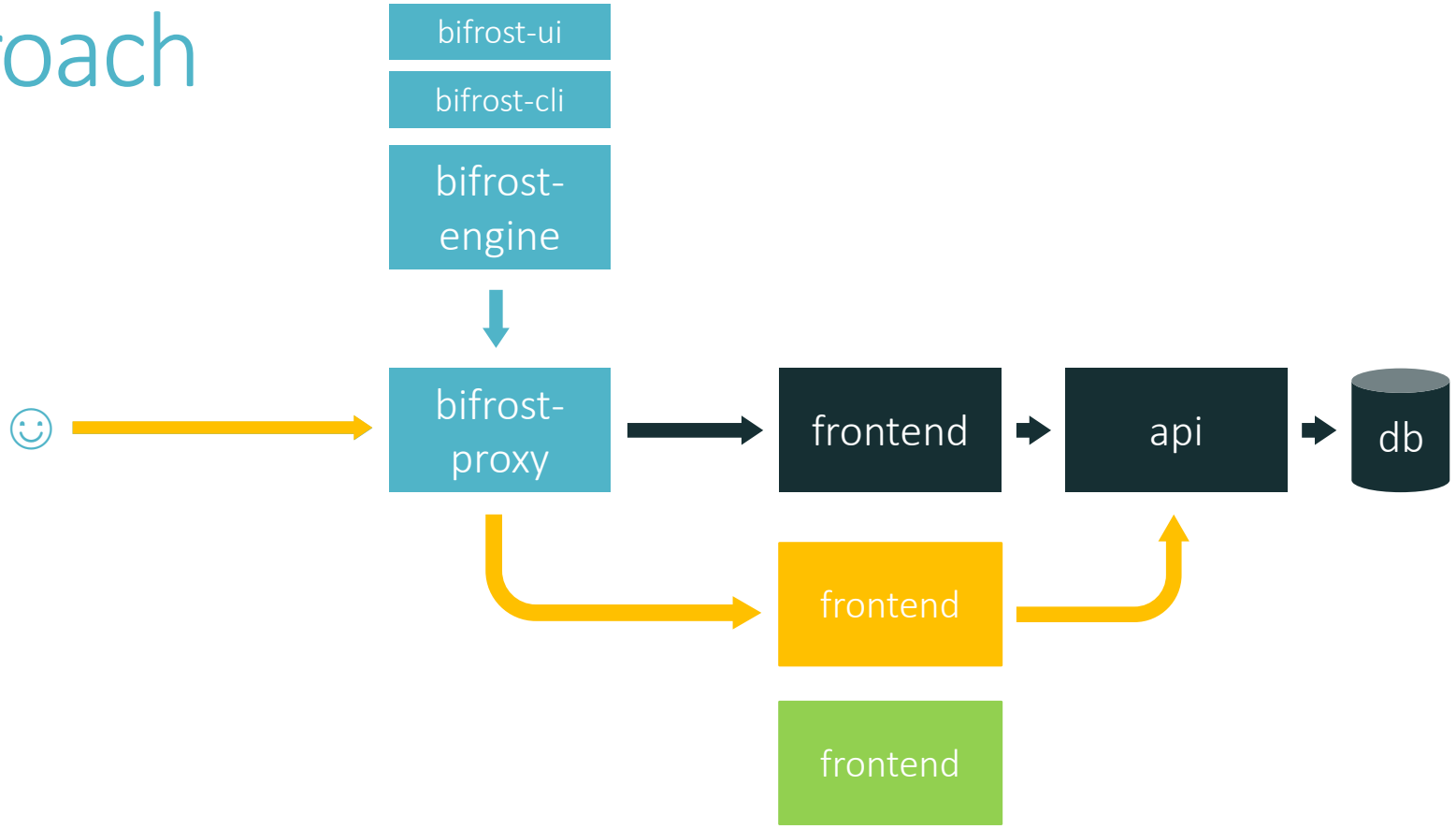
Feature Toggles

```
1  // Get all of a user's enabled features
2  var Features = fflip.userFeatures(someFreeUser);
3  if(Features.closedBeta) {
4    console.log('Welcome to the Closed Beta!');
5  }
6  // Or, just get a single one
7  if (fflip.userHasFeature(someFreeUser, 'closedBeta')) {
8    console.log('Welcome to the Closed Beta!');
9  }
```

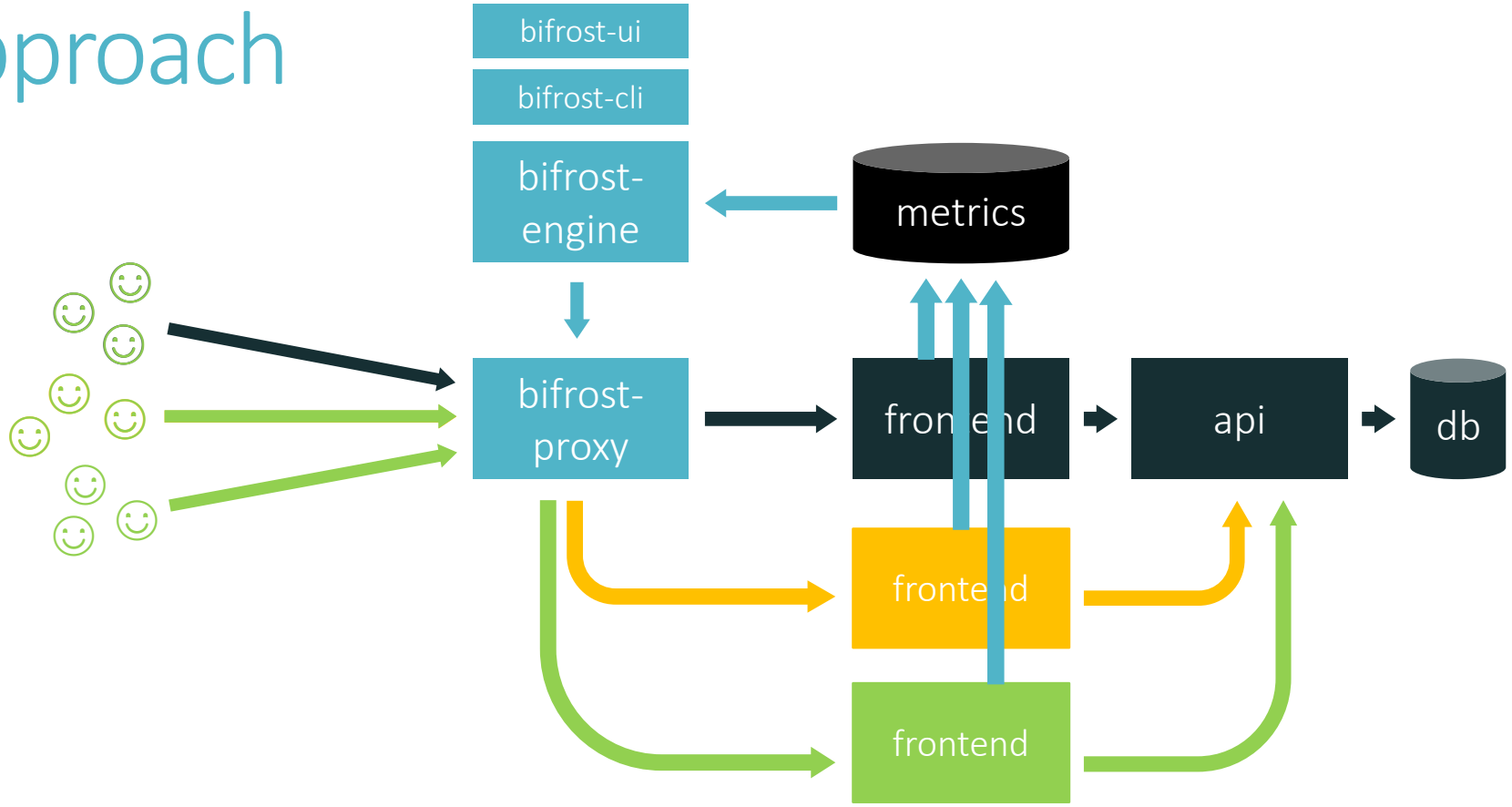
Dynamic Request Routing



Approach


bifrost-ui
bifrost-cli
bifrost-engine
bifrost-proxy
frontend
api
db
frontend
frontend

Approach


bifrost-ui
bifrost-cli
bifrost-engine
metrics
bifrost-proxy
frontend
api
db
frontend
frontend

Demo

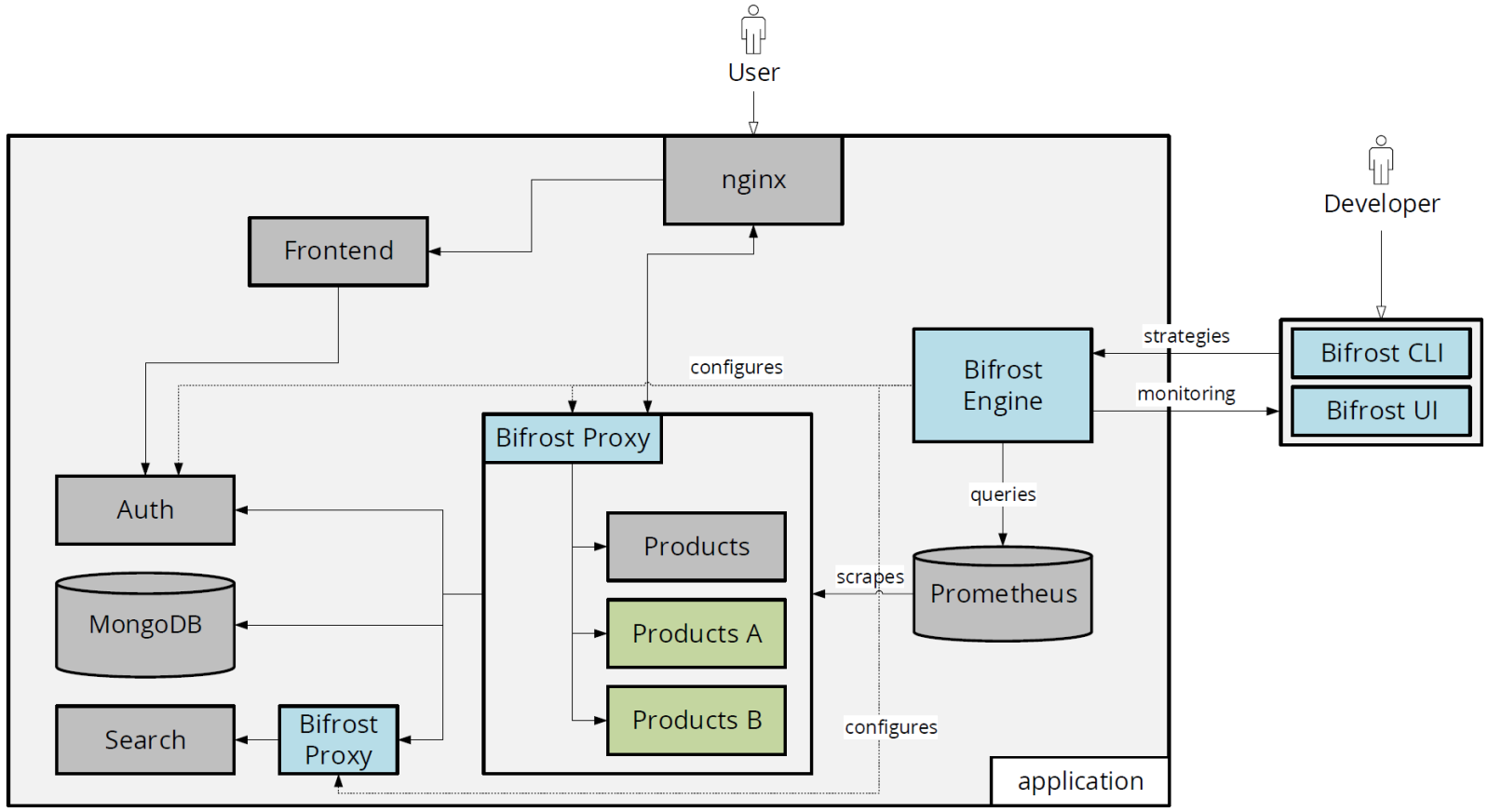
Evaluation

Qualitative Evaluation

Compared the **Bifrost Toolkit** to 5 similar tools using a set of defined dimensions.

- Tools exist, but mostly focus on specific use cases
- Most tools are either closed-source, tightly coupled with existing PaaS or locked-in to specific development platforms and architectures
- Bifrost Toolkit allows for more complex release procedures
- Only one additional tool allows flexible data-driven decision making

Quantitative Evaluation



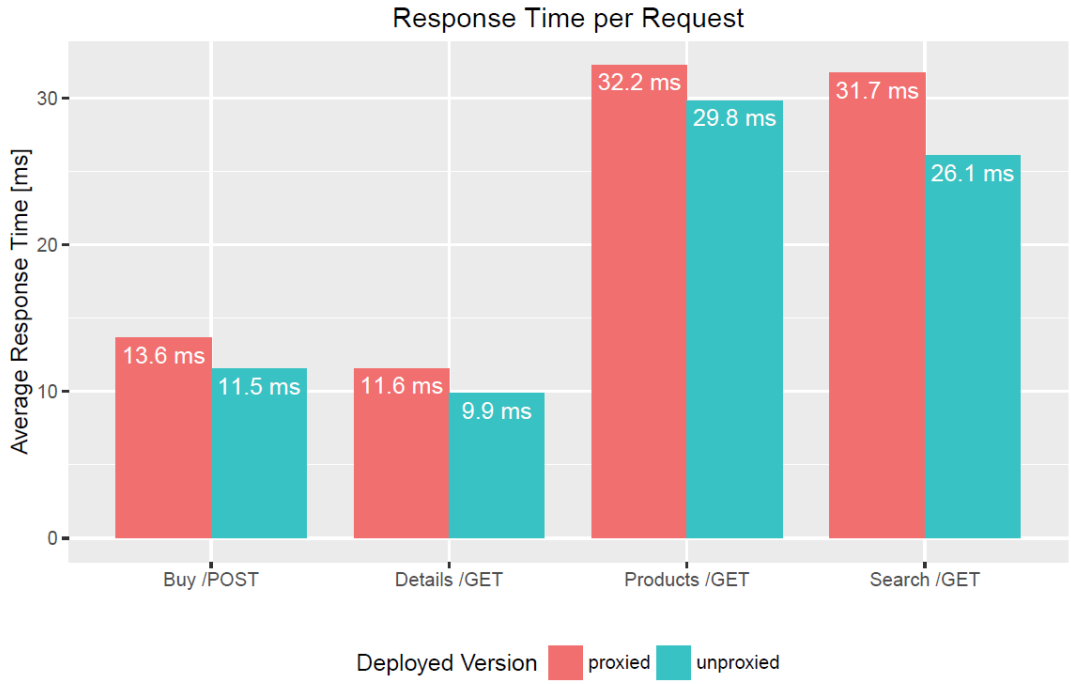
Request Performance

JMeter Testsuite fires 4 requests simulating normal user traffic to the application. Conducted with the Bifrost Toolkit deployed (**proxied**) and without (**unxproxied**), to measure the raw proxy overhead:

- Buying a product
- Querying product-details
- Wildcard-search for a product
- Retrieving list of all products

Request Performance

- One proxy instance adds approximately 2.4ms of delay
- The size of request and response can influence the added delay
- When deploying multiple proxies into the application landscape, a linear increase of the delay is to be expected



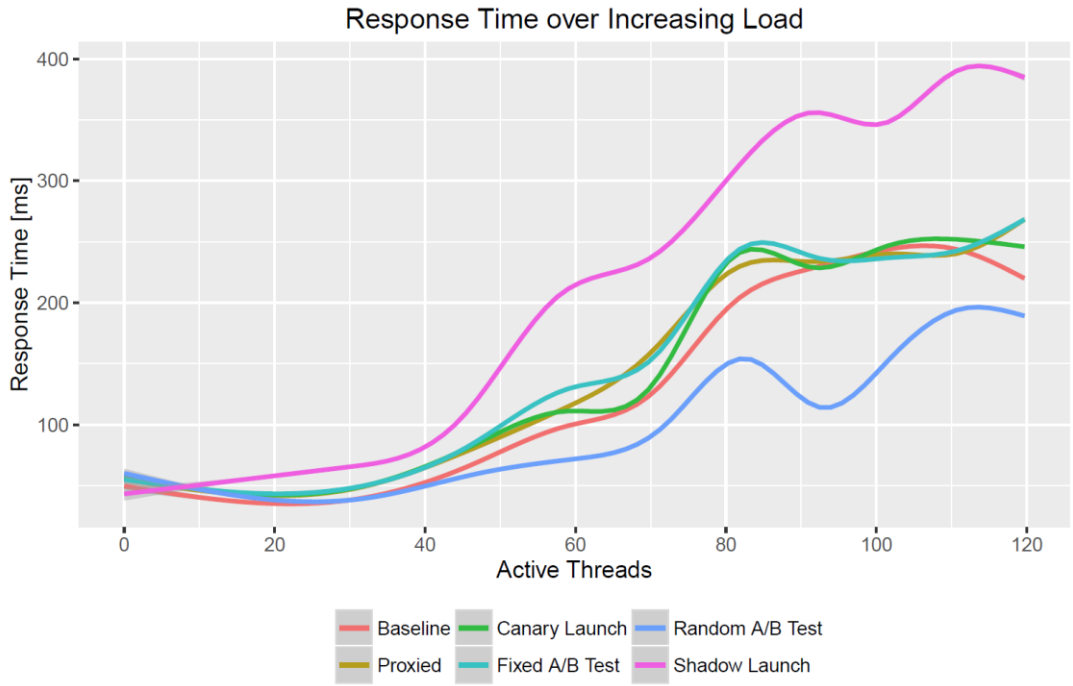
Filter Performance

4 different strategies/actions that simulate various live-testing methods under increasing load:

- A/B testing with fixed user allocation
- A/B testing with random user allocation
- Canary launching using header-filtering
- Shadow launching with traffic duplication

Filter Performance

- Fixed A/B testing adds a performance overhead
- Canary launches (and thus header filtering) show similar performance as running the proxy without filter
- Shadow launches increased the response time noticeably.



Release Performance

Simulated a complete release using multiple live-testing methods:

Canary Launch

- 5% Traffic Redirection each to Product A and Product B for 60 seconds.

Shadow Launch

- Duplicate Traffic to Product A and Product B for 60 seconds.

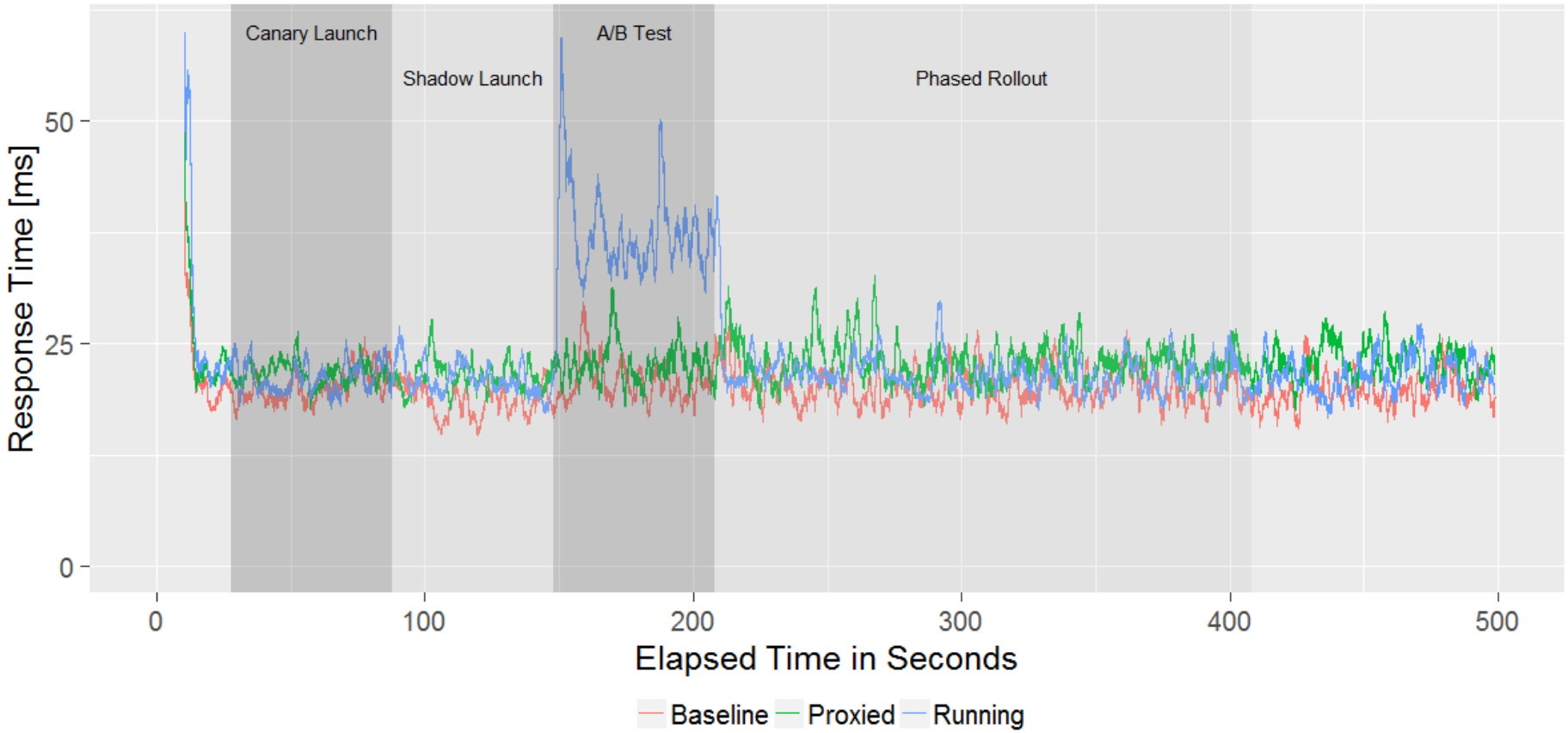
A/B Test

- Monitor sales performance of Product A and Product B using sticky sessions for 60 seconds.

Phased Rollout

- Rollout winner of A/B test over 200 seconds.

Release Performance



Summary and Conclusion

«How can we formalize a (generic) model for data-driven release and deployment strategies?»

Introduces a generic model for data-driven release and deployment strategies.

- Formalized using a mathematical model.
- Built using a usage scenario, incorporating identified characteristics of live-testing methods.

Summary and Conclusion

«How can we build a tool that supports and automates data-driven release and deployment strategies for microservices-based architectures in a non-intrusive way?»

Implemented a prototype using dynamic request routing, that makes use of the Bifrost Release Model.

- Provides a wider range of functionality than existing tools and approaches.
- Is platform-independent and does not require any modification of existing source code.
- Introduces a measurable but small delay into applications, and thus could profit from further optimizations.

Future Work

Possible improvements:

- Introduce formal model verification
- Add additional metric-providers
- Explore detailed integration in deployment pipeline
- Support for feature toggles
- Provide a deeper integration with existing IaaS- or PaaS-providers

Questions?

Bifrost Release Model

$$Release: \quad \{\{c_1, \ldots, c_n\}, (s_1, \ldots, s_n)\}$$

$$Strategy: \quad \{Parallel_A, Success_A, Failure_A\}$$

$$Parallel_A: \quad \{a_1, \ldots, a_n \mid a_i \in Action\}$$

$$Success_A: \quad \{x \mid x \in Action\}$$

$$Failure_A: \quad \{x \mid x \in Action\}$$

$$Action: \quad A = \{\Theta, \Delta, \Omega\} \qquad f_a(a_i) = result \mid result \in \{True, False\}$$

$$\Theta: \quad f(c_i) \rightarrow c_i'$$

Bifrost Release Model

Evaluation of Action using a Timer

$$f_{\Delta}(a_i) = \{f_a(a_i)_1 ... f_a(a_i)_n\} = \begin{cases} True & |f_a(a_i) \in True| \geq \Delta_{Threshold} \\ False & |f_a(a_i) \in True| < \Delta_{Threshold} \end{cases}$$

Evaluation of Strategy:

$$f_s(\{a_1, .., a_n\}) \rightarrow f_a(a_1) \wedge ... \wedge f_a(a_n) \rightarrow result \mid result \in \{True, False\}$$

Architecture and Technology

Written in ECMAScript 6 (JavaScript)

Actions

Route

- %-Traffic Filter
- Header-Field Filter
- Sticky Sessions

Request

- HTTP-Request
- Checks for statuscode
- Healthcheck

Nested Actions

- AND/OR Actions
- Nesting allows to model complex rulesets

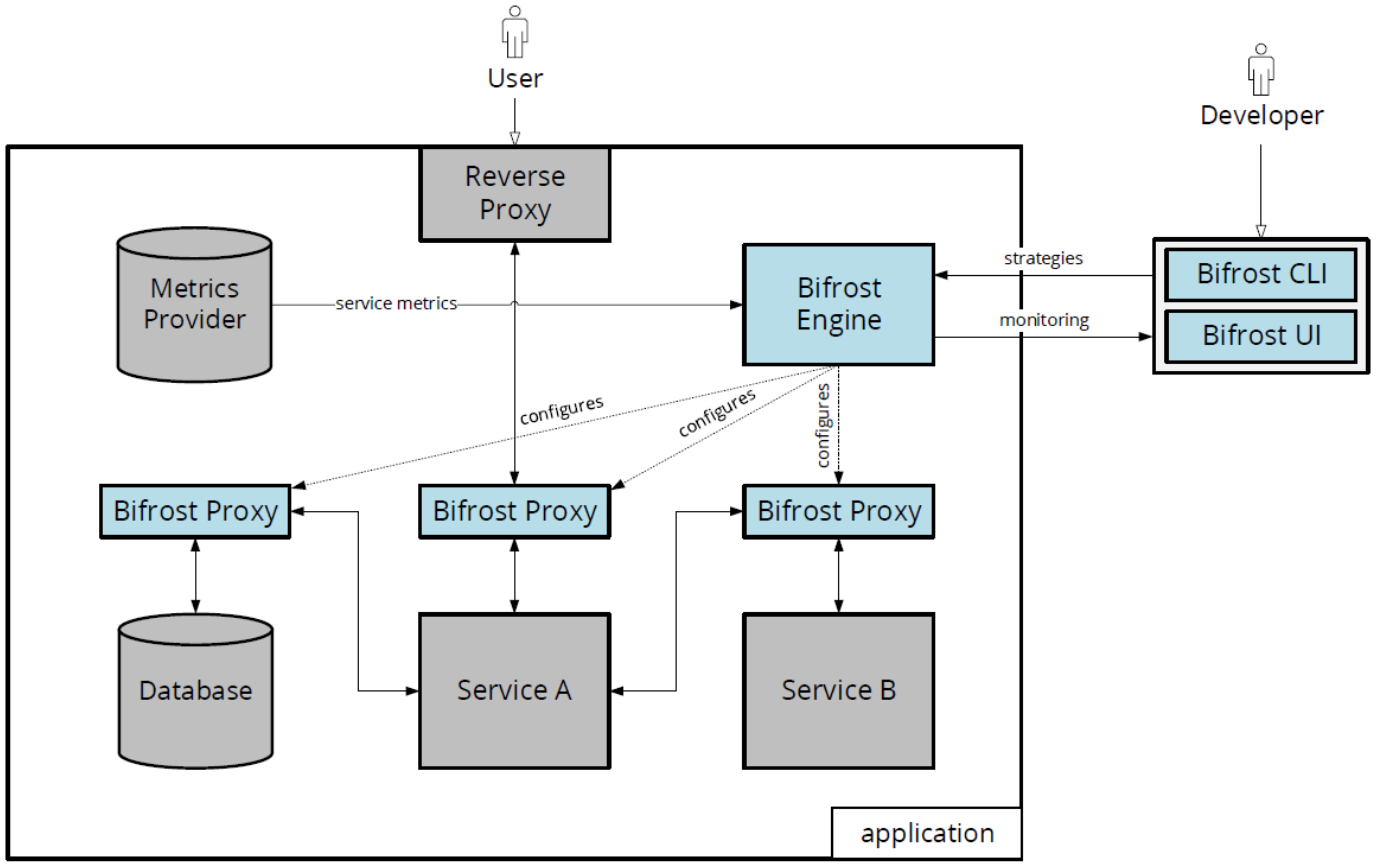
Stop

- Allows for manual confirmation after strategy

Metric

- Various providers possible.
- Simple inline-validation or custom validation using HTTP-Requests

Architecture



Challenges

Docker is changing their **network** feature. No support for new alias-feature on API yet.

- https://github.com/docker/docker/issues/18699

Compared Tools

	A/B Testing	Canary Launches	Shadow Launches	Phased Rollouts	Blue/Green Deployments	Combination
GateKeeper	Yes	Yes	Yes	Partial	No	Yes
CanaryAdvisor	No	Yes	No	No	No	Yes
Vamp	Yes	Yes	No	No	Yes	No
Scientist!	Partial	Partial	Yes	Partial	No	Partial
ION-Roller	No	Partial	No	Yes	Yes	No
Bifrost	Yes	Yes	Yes	Yes	Yes	Yes

Table 5.1: Feature-comparison of analyzed tools

Qualitative Evaluation - Results

	Platform-Agnostic	Code-Neutral	Performance-Neutrality	Traceability	Automated Data-Driven Decisions	Complex Releases	Open-Source
GateKeeper	Partial	No	Yes	Yes	Yes	Yes	No
CanaryAdvisor	Yes	Yes	Yes	No	Partial	No	No
Vamp	Partial	Yes	No	Yes	No	No	Yes
Scientist!	No	No	Partial	No	No	No	Yes
ION-Roller	Partial	Yes	Yes	Yes	No	No	Yes
Bifrost	Yes	Yes	No	Yes	Yes	Yes	Yes

Table 5.2: Comparison of live-testing tools

Request Performance

Request	Type	Mean	+/-	SD	Min	Max	Median
Buy /POST	Proxied	13.64	+2.5	16.82	5	252	8
	Unproxied	11.14		16.11	4	201	6
Details /GET	Proxied	11.58	+1.71	16.33	5	269	7
	Unproxied	9.9		16.33	4	215	6
Products /GET	Proxied	32.25	+2.45	20.82	14	281	27
	Unproxied	29.80		21.45	12	339	24
Search /GET	Proxied	31.69	+5.63	25.23	10	340	25
	Unproxied	26.06		23.89	7	267	19

Table 5.3: Results of request performance test in milliseconds

Example: Deployment/Services

```
---
name: Webshop Redesign A/B Test
deployment:
 orchestrator:
   proxy:
     mapping:
       frontend: bifrost_frontend_proxy

 services:
  - name: ClassicShop
    host: frontend
    port: 3000

  - name: RedesignedShop
    host: frontend_redesigned
    port: 3000
```

Example: Strategies

```
---
strategies:
 -  name: 50_50_traffic
    actions:
      - route:
          from: ClassicShop
          to: RedesignedShop
          filters:
            - traffic:
                percentage: 50
          intervalTime: 60
    next: check_metrics

 -  name: check_metrics
    actions:
        …
```

Example: Strategies

```
AND:
 actions:
  - OR:
      - metric:
          providers:
            - prometheus:
                query: "avg_over_time(container_cpu_usage_seconds_total{instance="workerA_1"}[60s])"
          validator: ">0.5"
      - metric:
          providers:
            - prometheus:
                query: "avg_over_time(container_cpu_usage_seconds_total{instance="workerA_2"}[60s])"
          validator: ">0.5"
   - OR:
      - metric:
          providers:
            - prometheus:
                query: "avg_over_time(container_cpu_usage_seconds_total{instance="workerB_1"}[60s])"
          validator: ">0.5"
      - metric:
          providers:
            - prometheus:
                query: "avg_over_time(container_cpu_usage_seconds_total{instance="workerB_2"}[60s])"
          validator: ">0.5"

onTrue: "highCPU"
onFalse: "lowCPU"
```

Approach: Specifying Release Strategies

```
---
name: Webshop Redesign A/B Test
deployment:
 orchestrator:
   proxy:
     mapping:
       frontend: bifrost_frontend_proxy

 services:
  - name: ClassicShop
    host: frontend
    port: 3000

  - name: RedesignedShop
    host: frontend_redesigned
    port: 3000
```

Approach: Specifying Release Strategies

```
strategies:
 -  name: 50_50_traffic
    actions:
      - route:
          from: ClassicShop
          to: RedesignedShop
          filters:
            - traffic:
                percentage: 50
          intervalTime: 60

    next: check_metrics

 -  name: check_metrics
    actions:
        …
```