PREvant (Preview Servant): Composing Microservices into Reviewable and Testable Applications

Marc Schreiber
aixigo AG, Aachen, Germany
marc.schreiber@fh-aachen.de

Abstract

This paper introduces PREvant (preview servant), a software tool which provides a simple RESTful API for deploying and composing containerized microservices as reviewable applications. PREvant's API serves as a connector between continuous delivery pipelines of microservices and the infrastructure that hosts the applications. Based on the REST API and a web interface developers and domain experts at aixigo AG developed quality assurance workflows that help to increase and maintain high microservice quality.

2012 ACM Subject Classification Software and its engineering → Software creation and management; Software and its engineering

Keywords and phrases Microservice development, testing for microservices, exploratory testing, development workflows

Digital Object Identifier 10.4230/OASIcs.Microservices.2017-2019.5

Supplement Material

- The source code is available on GitHub: `https://github.com/aixigo/PREvant`
- A short talk about PREvant is available on YouTube: `https://www.youtube.com/watch?v=O9GxapQR5bk`

1 Introduction

Currently, an increasing number of enterprises develop microservices to build their applications [2, 11, 18] because microservices are scalable and offer quick deployment cycles, superior quality, and greater flexibility compared to monolithic software [21, 24]. Furthermore, numerous companies migrate their on-premises applications to microservice architectures [13] because of the aforementioned advantages combined with agile software development. However, when development teams begin to build microservices, they can face major challenges due to the increased cognitive load, design complexity, testing and maintenance efforts [30].

Microservices must be deployed on an infrastructure to perform automatic and user-based acceptance tests, thereby ensuring feature correctness. Developers can employ containerization techniques to package and deploy their microservices [17]; however, they must manage the complexity of deployment set-ups when they provide the whole application as a preview, which consists of multiple microservices distributed across numerous source-code repositories. To deploy their services to a container orchestration platform for testing purposes, developers must create complex continuous delivery pipelines that utilize container orchestration platforms. In addition to these technical challenges, the testing of microservices provides further challenges that require effective testing strategies [15, 22].

This paper introduces *PREvant (Preview Servant)*, a software tool that helps to deliver high-quality microservices by providing an approach to deploying and composing containerized microservices as reviewable applications. PREvant simplifies the deployment of applications that comprise multiple microservices, and it supports various configuration scenarios that

Joint Post-proceedings of the First and Second International Conference on Microservices (Microservices 2017/2019).
Editors: Luís Cruz-Filipe, Saverio Giallorenzo, Fabrizio Montesi, Marco Peressotti, Florian Rademacher, and Sabine Sachweh; Article No. 5; pp. 5:1–5:16

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

equip the application with the necessary infrastructure through companions. Thanks to these and additional features, developers and domain experts can employ established workflows to ensure that their teams are building high-quality services. Additionally, sales and team managers can utilize PREvant's capabilities.

The remainder of this paper is structured as follows: Section 2 provides basic concepts and technologies that are necessary for understanding PREvant's use cases. Additionally, this section presents related work. Section 3 provides an exemplary case study to support PREvant's use cases with a meaningful example, and Section 4 depicts PREvant's approach, implementation, and architecture. Section 5 illustrates established workflows utilized at aixigo AG that ensure high quality microservices. Section 6 supplies recipes that have been collected through the utilization of PREvant at aixigo AG. Finally, Section 7 concludes the paper.

2 Background and Related Work

The development of microservices is based on the following characteristics [18]:

> Microservices are small, autonomous services that work together ... and [a service] might be deployed as an isolated service on a platform as a service (PAAS), or it might be its own operating system process. ... All communication between the services themselves are via network calls, to enforce separation between the services and avoid the perils of tight coupling. ... [The] service exposes an application programming interface (API), and collaborating services communicate with us via those APIs.

Therefore, developers who create applications consisting of independent microservices must ensure that the services provide feature correctness even when the independent microservices do not share any resources such as operating systems or databases. Due to the independence of microservices, Docker [17] and standard containerization techniques [9] support the development of such microservice architecture because Docker and containers in general suit each other in implementing this kind of architecture [14]. This approach of implementing microservice architectures [9, 14, 17] provides the foundation of PREvant's approach.

Because microservice architectures rely on unreliable network calls that require fault-tolerant services [18, Chapter 11], developers must ensure that the source code includes this fault tolerance. Therefore, developers can rely on automated unit tests and continuous integration to ensure high-quality code [12]. Additionally, developers must ensure that the whole application works as expected, a factor which is often confronted by continuous delivery pipelines [4]. Figure 1 depicts a continuous delivery pipeline for a microservice as described by Chen [4].

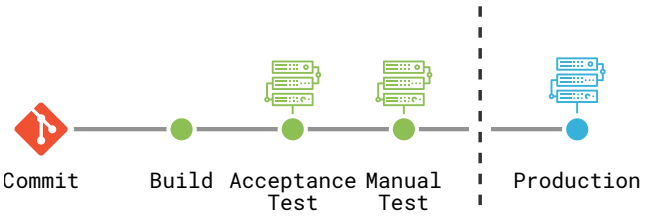

Figure 1 Continuous Delivery Pipeline Stages.

The build stage in Figure 1 represents the continuous integration stage, in which the microservice is compiled, tested, and packaged into a runnable format, such as a Docker container image [9, 17]. When the build stage is completed, the continuous delivery pipeline executes the next stages:

Acceptance Test In this stage, automated end-to-end tests ensure that the microservice performs as expected. For example, an automated browser test ensures that the microservice's data is rendered correctly.

Manual Test This stage provides a running instance of the microservice for domain experts who perform exploratory testing to ensure correct business behavior.

Production When the domain experts establish that the microservice performs correctly, the service can be deployed into production, completing the continuous delivery pipeline.

The acceptance and manual test stages include environments similar to the production environment in which the microservices will be deployed (depicted by the infrastructure symbols above the stages in Figure 1). The deployment in these environments is handled by the continuous delivery pipeline and is supported by different development tools, such as GitLab Review Apps[1] or GitOps,[2] that utilize container orchestrations to spin the required environments. Such an approach [4] builds common ground for GitLab Review Apps, GitOps, and PREvant.

However, existing solutions have a common disadvantage when developers build multiple microservices in multiple source-code repositories, a factor which is addressed by PREvant. Existing solutions such as GitLab Review Apps do not manage effectively when the development of an application consists of multiple microservices distributed across multiple source-code repositories, as depicted in Figure 2. In this scenario, the continuous delivery pipelines of each microservice repository (see the commit stage) must be aware of foreign microservices and ensure that their environments include them (see question marks in Figure 2). Therefore, both GitLab Review Apps and GitOps offer to write deployment scripts that ensure that the foreign microservices are deployed as well, which generates a tight coupling between the deployment pipelines, thus violating the mantra of independent microservice architecture.

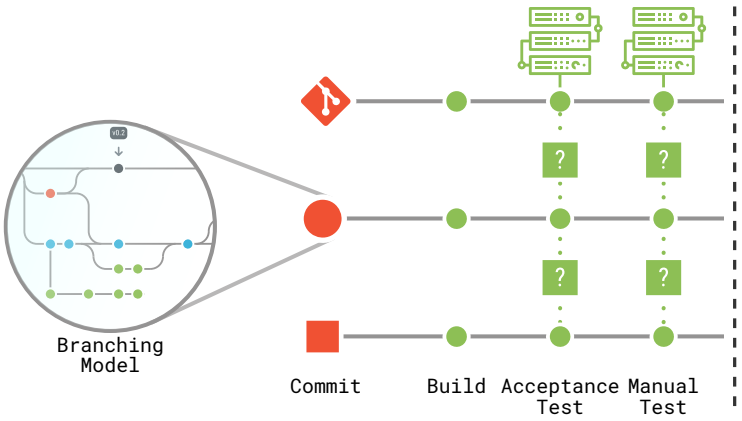

Figure 2 Continuous Delivery Pipeline Stages with Multi-Repository Development.

[1] https://about.gitlab.com/product/review-apps/
[2] https://www.weave.works/technologies/gitops/

Furthermore, the development of each microservice could follow a branching model, as depicted on the left-hand side of Figure 2, and each branch should be tested through the same stages of the delivery pipeline as the mainline branch. However, if a feature requires the extensions of two or more microservices, then this must be configured and written into the source code of the deployment scripts, which requires thorough clean-up afterward. This clean-up could fail and break the deployment of the mainline branches.

In contrast to existing solutions, PREvant aims to improve the handling of delivery pipelines in multi-repository, multi-branch scenarios, some of which are described in further detail in Section 4. Therefore, PREvant relies on following techniques to compose a set of microservices into one application:

1. Because PREvant supports the development of microservices, and containers are an ideal match for microservice architecture [14], PREvant relies on container runtime infrastructure, such as Docker, Docker Swarm, or Kubernetes to deploy the microservices into staging areas.[3]
2. To compose the microservices on a container runtime, PREvant relies on a container registry to exchange the container images between a continuous delivery pipeline and the container runtime.
3. Additionally, the container runtimes provide software-defined networking [7] that PREvant utilizes to isolate the microservice applications from each other.
4. To make the composed applications accessible to the domain experts, PREvant equips the containers in such a way that Traefik [5] can work as a reverse proxy, which makes Traefik a requirement for PREvant.

3 Case Study

To support PREvant's use cases with an illustrative example, this section introduces a sample e-commerce system that offers end customers the ability to purchase products in a web shop. The example is borrowed from Wolff [29] and provides the following services:

order This microservice provides a web interface that accepts orders through a shopping cart. All accepted orders are stored in a database and are published through an asynchronous messaging channel.

invoice This service subscribes to the orders channel and extracts all relevant information from the messages. The relevant information is stored as new invoices in a database, and the accounting department receives new invoices through the invoice service's web interface.

shipping Similar to the invoice service, the shipping service subscribes to the orders channel and stores the relevant shipping information in a database. The shipping department receives new shipping requests through the shipping service's web interface.

In the context of this paper, the development of these services is distributed across three source-code repositories, and each repository manages a continuous delivery pipeline that ensures that the services perform as desired. Additionally, the development team is supported through domain experts who randomly perform exploratory testing on the whole e-commerce system to ensure that the system works well for the salespeople who operate the system.

[3] Currently, PREvant only supports Docker, but PREvant's developers plan to support Kubernetes in the future.

Therefore, the continuous delivery pipelines must ensure that domain experts can access the application at any time, as depicted in Figure 2.

Furthermore, the microservices rely on infrastructure services. They require a Apache Kafka[4] instance to exchange the order message and PostgreSQL[5] as a database instance to store the service-relevant information. These infrastructure services must be deployed through the continuous delivery pipelines as well to ensure a running application.

4 Composing Microservices with PREvant

As stated in Section 2, PREvant aims to simplify the composition of microservices through continuous delivery pipelines. Therefore, PREvant serves as a connector between the continuous delivery pipelines and the infrastructure that hosts the applications for testing and quality-assurance purposes, as depicted in Figure 3. This approach facilitates the composition of reviewable applications that are explained in detail in this section.

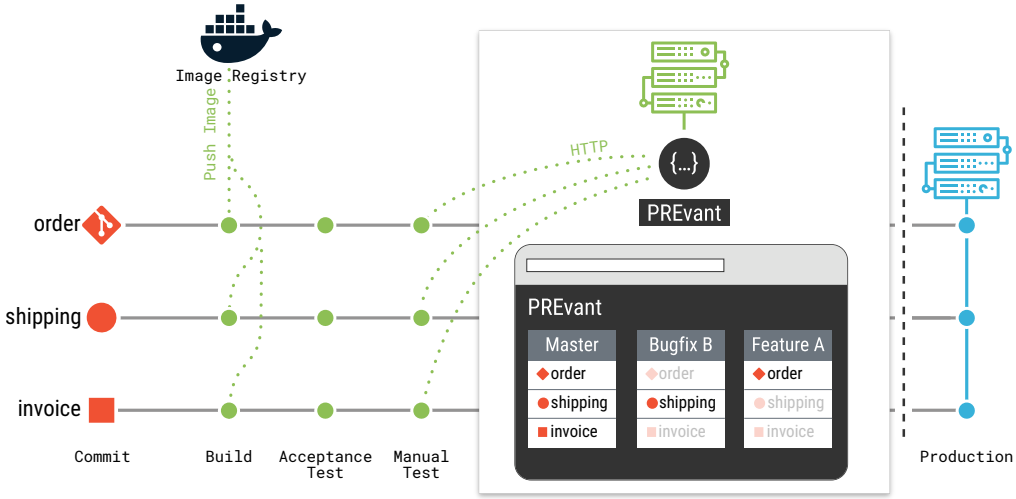

Figure 3 Composing Microservices with PREvant into Reviewable Applications.

Figure 3 illustrates the disjoint repositories and continuous delivery pipelines of the microservices *order*, *shipping*, and *invoice*. The build stage packages the microservices as a container image and pushes it to a container image registry (e.g. a Docker registry) to ensure that the services are ready for deployment in the acceptance and manual test stages. In a deployment phase, such as the manual test stage depicted in Figure 3, the continuous delivery pipeline can utilize PREvant's REST API, as illustrated in Listing 1. This REST request creates a software-defined network [7], initiates the container for the microservice, connects it to the network, and creates a reverse-proxy configuration, making the service accessible through PREvant's web interface.[6] Subsequent REST calls check whether the container image has a newer version, and if so, then the container is updated.

[4] `https://kafka.apache.org/`
[5] `https://www.postgresql.org/`
[6] In this example the invoice service is available through the relative URL `/master/invoice`.

Listing 1 Deploy `"invoice"` Service.

```
POST /api/apps/master HTTP/1.1
Content-Type: application/json
Accept: application/json

[{
  "serviceName": "invoice",
  "image": "registry.example.com/a-team/invoice:master",
}]
```

When the continuous delivery pipelines of the remaining microservices utilize a similar REST call, for example, replacing `invoice` in Listing 1 with `order` or `shipping` PREvant initiates the containers and connects them to the existing software-defined network so that the services can communicate with each other. Then, domain experts can access the services through PREvant's web interface, and they can begin exploratory testing before the microservices are deployed to production. Through this approach, PREvant offers the following key concepts:

- PREvant's REST API does not expose the internal workings of the underlying container runtime, which makes the deployment infrastructure agnostic. By design, the software architecture of PREvant employs an infrastructure abstraction so that Docker and Kubernetes are supported platforms; however, PREvant is not limited to these container orchestration platforms. Section 4.1 provides an architectural overview.
- The REST API approach reduces the complexity of continuous delivery pipelines because the necessity of deployment scripts for specific container orchestration is eliminated.
- The REST API approach enables further use cases that are employed in development workflows, as illustrated in Section 5.
- PREvant supports different microservice architectures so that it is not limited to a specific kind of microservice architecture. This is implemented through the concept of companions, as explained in Section 6.

In addition to these key concepts, PREvant supports feature branch workflows across multiple build pipelines and is not limited to mainline branches of microservices. As illustrated in Figure 2 in Section 2, a feature-based branching model raises the following issue: how does one deploy or compose the whole application automatically when developing a new feature on a branch for a single microservice? For this use case, PREvant provides following solution:

While PREvant can compose the mainline branches of multiple microservices into one application, it also utilizes the mainline branch as a template for each feature branch, as illustrated in Figure 4. To provide a fully functional application that can be reviewed by the domain experts, PREvant replicates all missing services from the master application. For example, if the delivery pipeline executes a POST request to deploy the service `order`, then PREvant compares the set of running microservices in the master application with the provided set of microservices. Then it includes the microservices that are not included and initiates them to provide a fully functional application. Here, it would deploy new container instances of the container images for the services `shipping` and `invoice`.

To distinguish the applications, the deployment pipeline must choose the names that are defined by a path parameter at the REST API level, as illustrated in Listing 2. Here, the application named `feature-a` (see path parameter) is deployed with the container image of the `order` service that has been labeled with `feature-a`. These names can be derived from the branch names through the continuous delivery pipeline, which is an established pattern at aixigo AG.

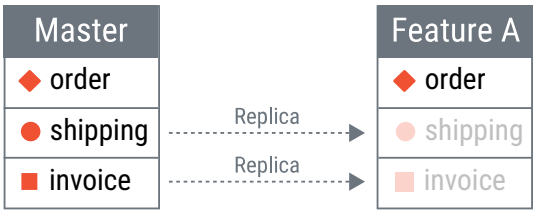

Figure 4 Replication of Services for Feature Branch Workflows.

Listing 2 Deploy Feature Branch of `"order"` Service.

```
POST /api/apps/feature-a HTTP/1.1
Content-Type: application/json
Accept: application/json

[{
  "serviceName": "order",
  "image": "registry.example.com/a-team/order:feature-a",
}]
```

Additionally, if a feature requires changes in two services, then the feature branches can use the same application names to deploy and test the feature across multiple microservice.

4.1 Architecture

To provide the aforementioned use cases, PREvant is implemented as a self-contained system [28] written in Rust [16]; it works in conjunction with Traefik and a container runtime, as illustrated in Figure 5. PREvant's architecture is divided into the following layers.

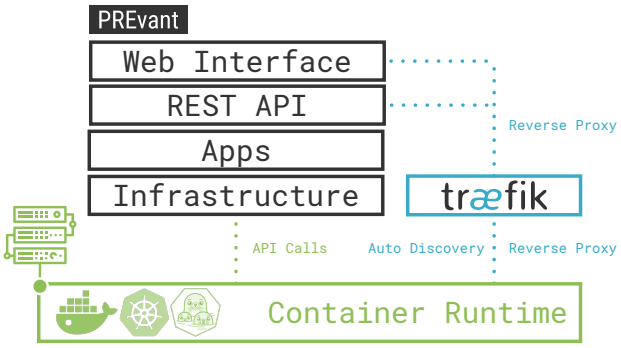

Figure 5 PREvant's Architecture.

Web Interface To provide the use case so that domain experts can access the reviewable applications, PREvant offers a web interface that employs PREvant's REST API to render the available applications. This interface is implemented as a single-page application written in Vue.js,[7] and Figure 6 displays the result in a screenshot of the web interface.

[7] `https://vuejs.org/`

REST API As an interface for continuous delivery pipelines and the web interface, PREvant implements its REST API with Rocket,[8] which is a web framework for Rust. This API layer forwards HTTP requests to the Apps layer.

Apps This layer implements the logic of PREvant's use cases. For example, the request to deploy a microservice is enriched with further information, such as additional services that must be deployed (see Figure 4), and the enriched information is passed to the Infrastructure layer.

Infrastructure The Infrastructure layer serves as a connector between the actual container runtime and the Apps layer by translating requests from the Apps layer into API calls to the container runtime. For example, when PREvant utilizes Docker as a container runtime, the request to list all running applications with the running services is translated into the corresponding API call,[9] as illustrated through *API Calls* in Figure 5. Further requests from the Apps layer are translated as well.

Additionally, this layer is responsible to create the software-defined network for every application so that the microservices of one application can communicate with each other. Additionally, this layer assigns DNS names to the services that are equivalent to the value of the field `"serviceName"` in the REST request.

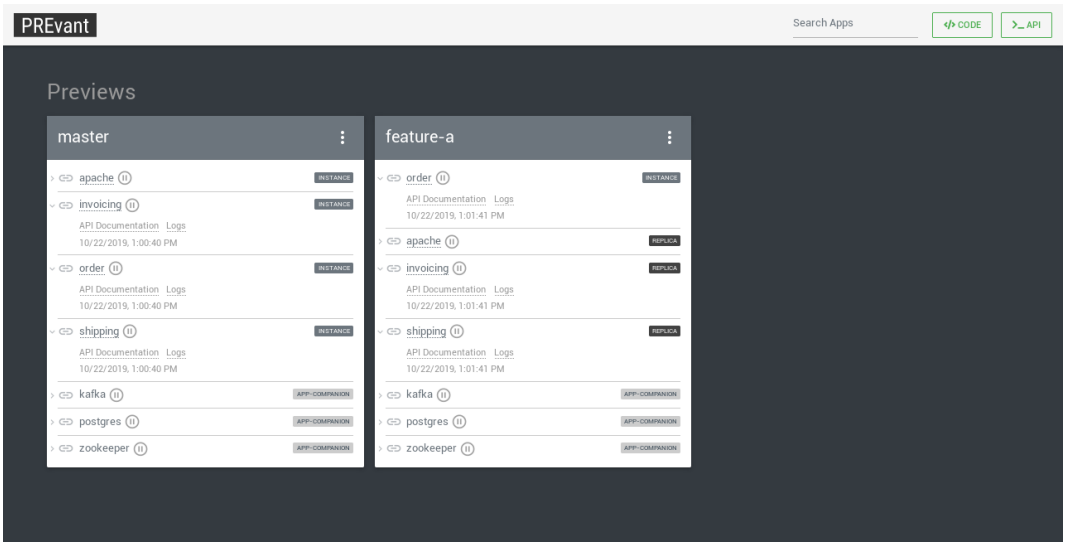

Figure 6 PREvant's Web Interface.

While this architecture solves the deployment and composition problems, it does not serve a proxy mechanism to make the microservices accessible. Therefore, PREvant utilizes Traefik's capabilities [5] as a reverse proxy. PREvant configures the running containers through the Infrastructure layer in such a way that Traefik can utilize the automatic discovery of containers to provide HTTP routes to the microservices, as illustrated in Figure 7. The HTTP requests of each web client, such as a browser, are routed through Traefik, which determines the microservice (concurrently running on the infrastructure) responsible to process the request. Additionally, Traefik routes all requests to PREvant's API and web interface.

[8] `https://rocket.rs/`
[9] `https://docs.docker.com/engine/api/v1.40/`

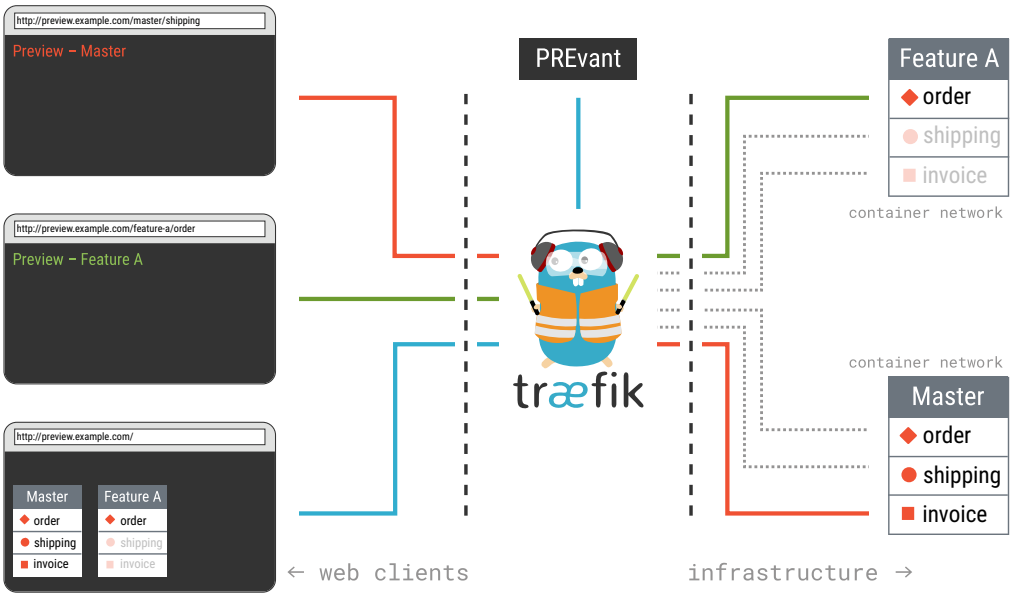

Figure 7 Traefik Serving the Microservices.

To utilize Traefik's capabilities, PREvant must label each microservice with routing information to enable Traefik to discover the microservices automatically. For example, if PREvant is hosted on `http://preview.example.com/`, then the `invoice` service of the application `master` would be configured to be accessible at following URL, as shown in Figure 7: `http://preview.example.com/master/invoice`

4.2 Conventions on Microservices and Application

As illustrated above, PREvant provides a web interface that makes PREvant's applications accessible to the domain experts, sales and team managers, and developers, as illustrated in Figure 6 which displays the microservices of the fictitious e-commerce shop in Section 3. This interface provides following features:

- Access to the microservices via HTML links, which are accessible through Traefik
- Access to log statements (see Logs in Figure 6)
- Issue-tracking system information
- Version information, such as Git commit hash, semantic version, and build date and time
- Swagger UI integration (see API Documentation in Figure 6)
- Start and stop buttons to test the resilience of each microservice

To provide these features, PREvant relies on some conventions. For example, to make the logs available, the container must log to standard output, which is a common practice for cloud-native applications [26]. Additionally, PREvant attempts to link the names of the applications to issue tracking information.

To provide version information and Swagger UI integration, PREvant collects information from the microservice itself. Therefore, PREvant employs the web host metadata, proposed in RFC 6415 [6]. If a microservice provides the well-known resource `/.well-known/host-meta.json`, then PREvant can collect the required information and render it in the web interface. Listing 3 illustrates web host metadata of the microservice `shipping` in the application `master`, demonstrating that the OpenAPI specification [8] of the microservice is available at `http://preview.example.com/master/shipping/swagger.json`. When PREvant requests the

well-known resource, it provides the HTTP headers `Forwarded` [19] and `X-Forwarded-Prefix` to the microservice, enabling the microservice to generate public links, which are required by the web interface.

Listing 3 Microservices Properties Formulated in Web Host Meta.

```
{
  "properties": {
    "https://schema.org/softwareVersion": "0.9",
    "https://schema.org/dateModified": "2019-08-12T15:31:00Z",
    "https://git-scm.com/docs/git-commit":
       "43de4c6edf3c7ed93cdf8983f1ea7d73115176cc"
  },
  "links": [
    {
      "rel": "https://github.com/OAI/OpenAPI-Specification",
      "href": "http://preview.example.com/master/shipping/swagger.json"
    }
  ]
}
```

5 Development and Quality Assurance Workflows

Aixigo AG utilizes PREvant extensively in daily development activities and has developed substantial workflows that improve the quality assurance of its microservices:

- Aixigo AG utilizes feature branches to prevent the pollution of mainline branches with incomplete features, as discussed in Section 5.1.
- Aixigo's microservices provide flexible configuration capabilities, and PREvant allows changes in configuration set-ups quickly to test different scenarios, as illustrated in Section 5.2.
- Aixigo AG provides bug fixes for older major or minor releases of its microservices. In these cases, PREvant helps to reproduce reported bugs, as outlined in Section 5.3.

Each described workflow occurs before the release to production, because aixigo AG generally cannot access the production area due to legal constraints. Some testing strategies of microservices suggest testing in the production area [15] but this paper does not cover such strategies; rather, it describes workflows that allow developers, domain experts, and sales and team managers to provide improved service quality for in-house developments.

5.1 Feature Branch Based Development

New feature development of a microservice at aixigo AG is subject to strict quality control because several actions have been implemented to ensure high quality: test-driven development [1], code reviews [3], pair programming [27], snapshot and integration tests, end-to-end tests.[10] Additionally, the company has enhanced its development workflow based on reviews by its domain experts.

When a new feature is completed by a developer, such as new functionality in the front-end or a new REST resource, it must be reviewed by a domain expert; the domain experts have

[10] Vocke [25] provides more information about automatic software tests.

experience with REST APIs, so they can judge whether the developer has implemented the feature correctly. When the feature branch of any microservice has been built by the delivery pipeline and the whole application is available on PREvant, as mentioned in Figure 4, the developer notifies the domain expert that the branch is ready for review, and the domain expert performs exploratory tests. If any issues are discovered, then they are reported to the developer; the developer then solves the reported issues.

Even when developers are working on features that are not observable from the outside, PREvant ensures that the application meets quality criteria. For example, aixigo AG's delivery pipelines deploy the feature branches for automatic end-to-end tests (see the acceptance stage). PREvant deploys the whole application and the end-to-end tests are working on the whole application, which ensures that the whole set of microservices is working in conjunction, thus reducing integration time and costs. Due to the quick deployment cycles that PREvant provides, teams at aixigo AG have extended their definitions of done with the clause that the developer is responsible to test the new application features manually (by clicking through the front-end or by utilizing the integrated Swagger UI), which prevents obvious faults by the developer.

When the feature is completed and complies with the definition of done[23, Page 18], the developer merges the feature onto the mainline branch. This invokes a web hook and instructs PREvant to shut down the application, and the development continues with the next feature.

5.2 Configurable and Isolated Environments

Because the microservices of aixigo AG are utilized with different configurations in different production environments, it is crucial that these configurations can be tested in an accessible manner. Therefore, the company utilizes PREvant's REST API to spin up-and-down applications with specific configuration scenarios. Listing 4 illustrates the usage of PREvant's REST API to spin up the application `features.test` with the `order` service in a specific configuration.

Listing 4 REST API Call with Configuration for Test Case.

```
POST /api/apps/features.test HTTP/1.1
Host: preview.example.com
Content-Type: application/json
Accept: application/json

[{
  "serviceName": "order",
  "image": "registry.example.com/a-team/order",
  "files": {
    "/etc/order/conf.d/sales.properties": "some.feature.flag = OFF"
  },
  "env": {
    "FEATURE_FLAG": "ON"
  }
}]
```

In this case the configuration of `order` is influenced by two parameters: the configuration file `/etc/order/conf.d/sales.properties` and the environment variable `FEATURE_FLAG` that toggle features of the service. The configuration of the remaining services has not been

changed. When the REST request has been executed, the domain experts and developers can test the service to determine whether it behaves correctly. This feature is crucial to quality assurance when introducing new configuration options because the mainline application runs in a default configuration; otherwise, it would be difficult for domain experts and developers to spin up an application for quality-assurance purposes. PREvant's approach provides additional use cases:

- The microservices with the specific configurations are running in isolation so that domain experts can test the application without disruption from builds and deployments that are executed on the mainline branch. Aixigo AG's domain experts often utilize this feature to clone the mainline branch to test it with the default configuration without disruption. The REST call that is integrated into PREvant's web interface is illustrated in Listing 5. Furthermore, the REST interface provides the query parameter `replicateFrom` to specify which application should be replicated.

Listing 5 REST API Call Cloning Mainline.

```
POST /api/apps/features.test?replicateFrom=master HTTP/1.1
Host: preview.example.com
Content-Type: application/json
Accept: application/json

[]
```

- From time to time, sales managers wish to demonstrate the application to potential customers and must demonstrate the application in a specific configuration. Therefore, they ask aixigo's domain experts to set up an application that can demonstrate feature X or Y.

5.3 Version Picking

Microservices of aixigo AG are running in production in different major or minor versions, and users report bugs in different versions. To reproduce these bugs, domain experts and developers utilize PREvant to spin up an application that runs the specific version with an additional specific configuration of the service. Therefore, a domain expert can utilize the REST API to select a specific container image version, as illustrated in Listing 6.[11]

Listing 6 REST API Call with Version Picking.

```
POST /api/apps/is-it-working-with-v1?replicateFrom=master HTTP/1.1
Host: preview.example.com
Content-Type: application/json
Accept: application/json

[{
  "serviceName": "order",
  "image": "registry.example.com/a-team/order:1.0.2"
}]
```

[11] Sometimes it is useful to compare old and new versions of a microservice in aixigo's sprint reviews; on such occasions, it is convenient to spin it up with PREvant.

PREvant does not provide a user interface for this use case and the feature is only accessible utilizing command line tools or the integrated Swagger UI. However, PREvant's road map contains the extension of the web interface, so that spinning up applications with specific versions and configurations, as illustrated in Listing 4 and 6, is more effective.

6 Companions and Recipes

While PREvant enables workflows that ensure a higher quality of microservices, it also aims to be agnostic to the hosted types of microservices, which means that the type of microservices [10] in development are irrelevant to PREvant. Whether a team develops a function as a service or a self-contained system, the microservices merely need to be packaged as a container image. However, these microservices rely on services that provide some infrastructure, such as databases or OpenID [20] providers, as depicted in the case study in Section 3. To provide these infrastructure services, PREvant offers two types of companions that are infrastructure microservices that are available over the container network while the microservice applications are running. PREvant deploys companions automatically when it received a REST request.

Service Companions Some infrastructure services are specific to a given microservice or are logically owned by a microservice. For example, a database instance such as MariaDB[12] is required by a service, a memory cache such as Memcached[13] is required by another service, and all services require a sidecar proxy.

These infrastructure services must be available for the microservice, and PREvant initiates these services as soon as the dependent microservice spins up.

Application Companions Some infrastructure services must be available for all microservices of an application. For example, Apache Kafka provides a stream-processing platform to establish publish and subscribe messaging between services. Further examples include service discovery providers, API gateways, or OpenID providers.

These types of services are globally available and potentially required by all microservices of the application. Therefore, PREvant initiates these infrastructure services when the application spins up.

To initiate these infrastructure services, PREvant provides configuration options that are illustrated in the following subsections by a set of recipes. These recipes provide some configuration examples that help developers to compose their microservice applications through PREvant. Section 6.1 provides an example of the configuration of database services for all microservices of an application. Additionally, these configuration options provide some templating options so that configurations are dynamically adjusted.

6.1 Service Databases

As stated previously, microservices that are hosted by PREvant often require the provision of a database. Listing 7 illustrates a configuration that ensures initiation of a MariaDB database, which is a service companion.

[12] `https://mariadb.org/`
[13] `http://memcached.org/`

Listing 7 PREvant Configuration: Database Companion.

```
1 [companions.mariadb]
2 type = 'service'
3 image = 'docker.io/library/mariadb:10.3'
4 serviceName = '{{service.name}}-db'
5 env = [
6   'MYSQL_DATABASE={{service.name}}', 'MYSQL_USER={{service.name}}',
    ↪  'MYSQL_PASSWORD={{service.name}}'
7 ]
```

Initially, the companion name must be defined; in this case, the name is `mariadb` (see Line 1). Additionally, the companion type must be defined so that it is valid for every service (see Line 2). Furthermore, the container image as well as the service name must be specified (see Line 3), which results in the DNS name of the database (see Line 4). The configuration of the DNS name can be adjusted by Handlebars templating syntax.[14] In this case, the corresponding microservice name is utilized to derive the DNS name. For example, for the microservice with name `X`, the corresponding database DNS name is `X-db`. Additionally, the environment variables of the database companion are adjusted so that the companion creates a database with a default username and password.

6.2 API Gateway

Some microservice architectures require an API gateway that processes every request before they are forwarded to any microservice. To initiate an API gateway for each application, definitions of the type, image and service name for the companion are required, as illustrated in Listing 8 by `[companions.api-gateway]`.

Listing 8 PREvant Configuration Using An API Gateway.

```
[companions.api-gateway]
type = 'application'
image = 'registry.example.com/a-team/api-gateway:latest'
serviceName = 'api-gateway'

[companions.api-gateway.labels]
'traefik.frontend.rule' = 'PathPrefix:/{{application.name}}/'
'traefik.frontend.priority' = '10000'
```

Furthermore, all requests are routed through the API gateway; therefore, the entry link a domain expert employs to interact with the application is irrelevant. In Listing 8, the configuration section labeled `[companions.api-gateway.labels]` ensures that the default labeling of the API gateway, which is responsible for Traefik's automatic discovery, is overwritten. Here, every request to the application is routed to the API gateway because of the higher priority and the path-prefix rule.[15]

[14] `https://handlebarsjs.com/`
[15] More information about Traefik's configuration options are available at: `https://docs.traefik.io/basics/`

7 Summary

This paper presented the concepts and implementation details of the tool Preview Servant (PREvant), which enables developers and domain experts to perform quality-assurance tasks on their microservice applications. Therefore, PREvant provides a simple REST interface that allows developers to extend their microservices' delivery pipelines with PREvant's composing mechanism (see Section 4), which spins up fully functional applications that can be explored by domain experts. Based on the capabilities of the RESTful interface, aixigo's employees utilize workflows that increase and maintain the microservice quality (see Section 5). Furthermore, PREvant's approach is not tied to these established workflows; rather, it employs an approach that is independent from the microservice application architecture because PREvant enables developers to configure required infrastructure services, as described in Section 6.

Furthermore, PREvant's composition and configuration mechanisms that are integrated in the web interface, allow users to set up dedicated previews in minutes so that sales managers can utilize isolated applications to demonstrate the application to potential customers. PREvant has become a major factor in the quality-assurance process at aixigo AG.

References

1 Dave Astels. *Test Driven Development: A Practical Guide*. Prentice Hall Professional Technical Reference, July 2003.

2 Antonio Bucchiarone, Nicola Dragoni, Schahram Dustdar, Stephan T. Larsen, and Manuel Mazzara. From Monolithic to Microservices: An Experience Report from the Banking Domain. *IEEE Software*, 35(3):50–55, May 2018. doi:10.1109/MS.2018.2141026.

3 Giuliana Carullo. *Code Reviews 101: The Wisdom of Good Coding*. Giuliana Carullo, May 2019.

4 Lianping Chen. Continuous Delivery: Huge Benefits, but Challenges Too. *IEEE Software*, 32(2):50–54, March 2015. doi:10.1109/MS.2015.27.

5 Containous. Traefik: The Cloud Native Edge Router, 2019. Accessed: 2019-08-12. URL: https://traefik.io/.

6 B. Cook. Web Host Metadata. Technical report, Internet Engineering Task Force, November 2011. doi:10.17487/rfc6415.

7 Cosmin Costache, Octavian Machidon, Adrian Mladin, Florin Sandu, and Razvan Bocu. Software-defined networking of Linux containers. In *RoEduNet Conference 13th Edition: Networking in Education and Research Joint Event RENAM 8th Conference*, pages 1–4, September 2014. doi:10.1109/RoEduNet-RENAM.2014.6955310.

8 The Linux Foundation. OpenAPI Initiative, 2019. Accessed: 2019-11-08. URL: https://www.openapis.org/.

9 Silvery Fu, Jiangchuan Liu, Xiaowen Chu, and Yueming Hu. Toward a Standard Interface for Cloud Providers: The Container as the Narrow Waist. *IEEE Internet Computing*, 20:66–71, 2016. doi:10.1109/MIC.2016.25.

10 Martin Garriga. Towards a Taxonomy of Microservices Architectures. In *Software Engineering and Formal Methods*, pages 203–218. Springer International Publishing, February 2018. doi:10.1007/978-3-319-74781-1_15.

11 Anne Marie Glen. [DZone Research] Microservices Priorities and Trends, July 2018. Accessed: 2019-08-12. URL: https://dzone.com/articles/dzone-research-microservices-priorities-and-trends.

12 Jesper Holck and Niels Jørgensen. Continuous Integration and Quality Assurance: a case study of two open source projects. *Australasian Journal of Information Systems*, 11(1), November 2003. doi:10.3127/ajis.v11i1.145.

13 Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. Cloud Migration Research: A Systematic Review. *IEEE Transactions on Cloud Computing*, 1(2):142–157, 2013. doi:10.1109/tcc.2013.10.

14 David Jaramillo, Duy Nguyen, and Robert Smart. Leveraging microservices architecture by using Docker technology. In *SoutheastCon 2016*, pages 1–5, March 2016. doi:10.1109/SECON.2016.7506647.

15 Sheroy Marker. Test Strategy for Microservices, May 2018. Accessed: 2019-08-12. URL: https://www.gocd.org/2018/05/08/continuous-delivery-microservices-test-strategy/.

16 Nicholas D. Matsakis and Felix S. Klock, II. The Rust Language. In *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*, HILT '14, pages 103–104. ACM, 2014. doi:10.1145/2692956.2663188.

17 Dirk Merkel. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux Journal*, 2014(239):2, 2014.

18 Sam Newman. *Building Microservices*. O'Reilly Media, Inc., 1st edition, 2015.

19 A. Petersson and M. Nilsson. Forwarded HTTP Extension. Technical report, Internet Engineering Task Force, June 2014. doi:10.17487/rfc7239.

20 David Recordon and Drummond Reed. OpenID 2.0: A Platform for User-centric Identity Management. In *Proceedings of the Second ACM Workshop on Digital Identity Management*, DIM '06, pages 11–16, New York, NY, USA, 2006. ACM. doi:10.1145/1179529.1179532.

21 Cesar Saavedra. The State of Microservices Survey 2017 – Eight trends you need to know, December 2017. Accessed: 2019-08-12. URL: https://middlewareblog.redhat.com/2017/12/05/the-state-of-microservices-survey-2017-eight-trends-you-need-to-know/.

22 D. I. Savchenko, Gleb Radchenko, and Ossi Taipale. Microservices validation: Mjolnirr platform case study. In *38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 235–240, May 2015. doi:10.1109/MIPRO.2015.7160271.

23 Jeff Sutherland and Ken Schwaber. The scrum guide. *The definitive guide to scrum: The rules of the game*, 268, 2013. URL: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf.

24 Markos Viggiato, Ricardo Terra, Henrique Rocha, Marco Tulio Valente, and Eduardo Figueiredo. Microservices in Practice: A Survey Study. In *VEM 2018 - 6th Workshop on Software Visualization, Evolution and Maintenance*, Sao Carlos, Brazil, September 2018. URL: https://hal.inria.fr/hal-01944464.

25 Ham Vocke. The Practical Test Pyramid, February 2018. Accessed: 2019-11-08. URL: https://martinfowler.com/articles/practical-test-pyramid.html.

26 Adam Wiggins. The Twelve-Factor App, 2017. Accessed: 2019-08-12. URL: https://12factor.net/.

27 Laurie Williams. *Pair Programming Illuminated*. Addison-Wesley Professional, July 2002.

28 Eberhard Wolff. *Microservices: Flexible Software Architectures*. CreateSpace Independent Publishing Platform, 2016.

29 Eberhard Wolff. *Microservices: A Practical Guide*. CreateSpace Independent Publishing Platform, April 2018.

30 Olaf Zimmermann. Microservices Tenets. *Comput. Sci.*, 32(3–4):301–310, July 2017. doi:10.1007/s00450-016-0337-0.