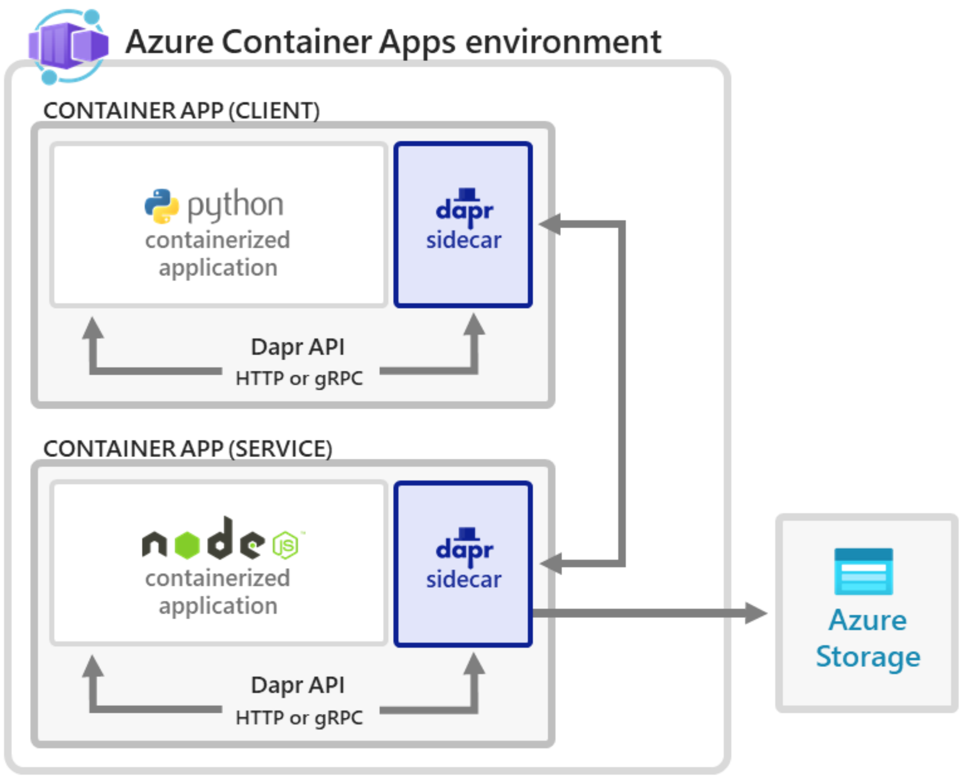
Dapr on Azure: A Practical Guide to Polyglot Microservice

Dapr (Distributed Application Runtime) is a an open-source, portable runtime that simplifies building resilient, microservice-based applications.



In the ever-evolving landscape of application development, microservices have emerged as a prominent architectural style. Microservices enable the development of large, complex applications by breaking them down into smaller, independent services. The use of multiple programming languages further enhances the flexibility and scalability of microservices.

This blog explores the integration of Java-Springboot, .NET, Go Lang, and Rust within a microservices architecture using Microsoft Azure and Dapr.

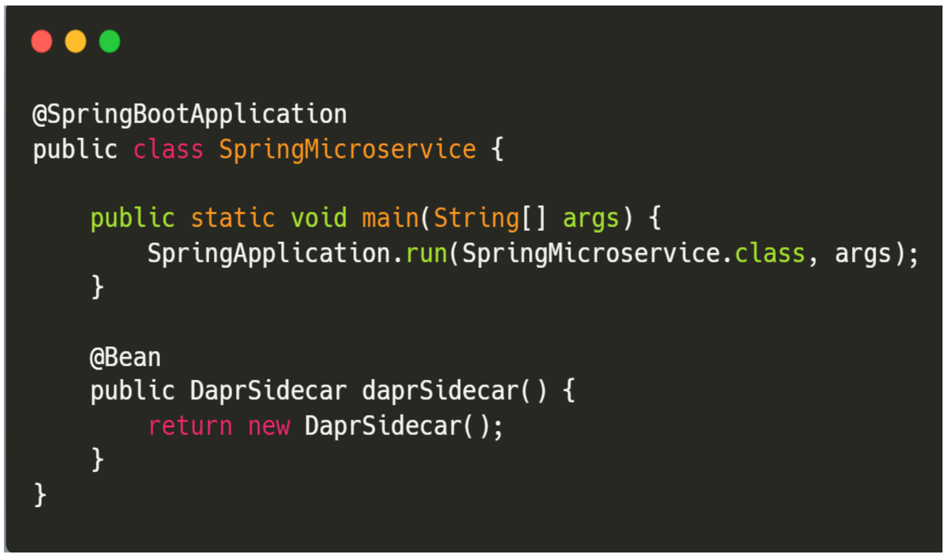
Problem Statement

Developers often face the challenge of integrating microservices written in different programming languages into a scalable system. This is particularly evident in scenarios where legacy systems need to interact with modern services or when different teams prefer different languages for their microservices.

Solution/ Architecture

To address this challenge, we can use the power of Dapr, an open-source, event-driven runtime that makes it easy to build resilient, scalable, and distributed systems. The architecture consists of microservices implemented in Java-Springboot, .NET, Go Lang, and Rust, communicating through Dapr.

Dapr Initialization:



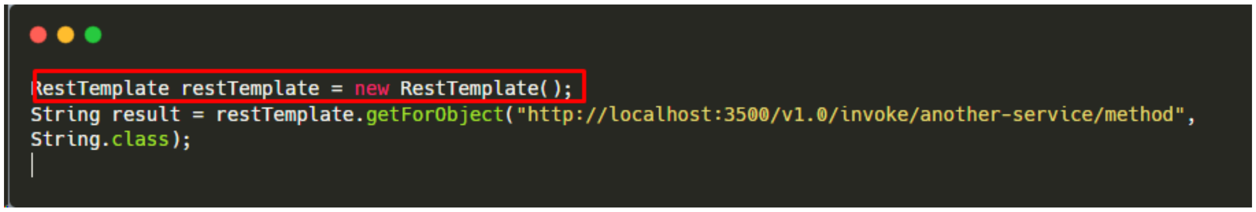
```
@SpringBootApplication
public class SpringMicroservice {

    public static void main(String[] args) {
        SpringApplication.run(SpringMicroservice.class, args);
    }

    @Bean
    public DaprSidecar daprSidecar() {
        return new DaprSidecar();
    }
}
```

Each microservice initializes Dapr to enable communication and service discovery. Above is an example of how Dapr is initialized in a Java-Springboot microservice.

Similarly, .NET, Go Lang, and Rust microservices have their own Dapr initialization code.

Service-to-Service Communication

Dapr provides a consistent API for service-to-service communication, regardless of the programming language. Here's an example of how a Java-Springboot microservice can call another microservice:



```
RestTemplate restTemplate = new RestTemplate();
String result = restTemplate.getForObject("http://localhost:3500/v1.0/invoke/another-service/method",
String.class);
```

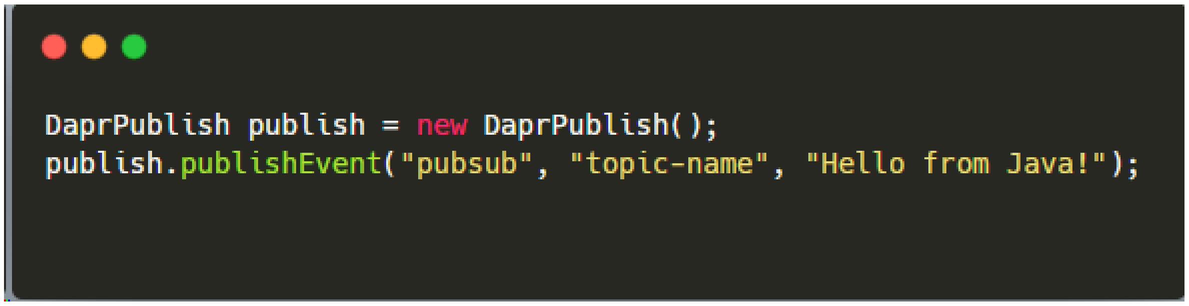
This approach is replicated in .NET, Go Lang, and Rust microservices, ensuring a uniform communication model.

Domain Segregation

To achieve domain segregation, each microservice focuses on a specific business domain. For example, the Java-Springboot microservice may handle user authentication, the .NET microservice manages order processing, the Go Lang microservice handles notifications, and the Rust microservice deals with inventory management.

Dapr Implementation

Dapr facilitates communication through its building blocks like state management, pub/sub, and service invocation. Below is an example of using Dapr's pub/sub feature to send a message from the Java-Springboot microservice to the Go Lang microservice:



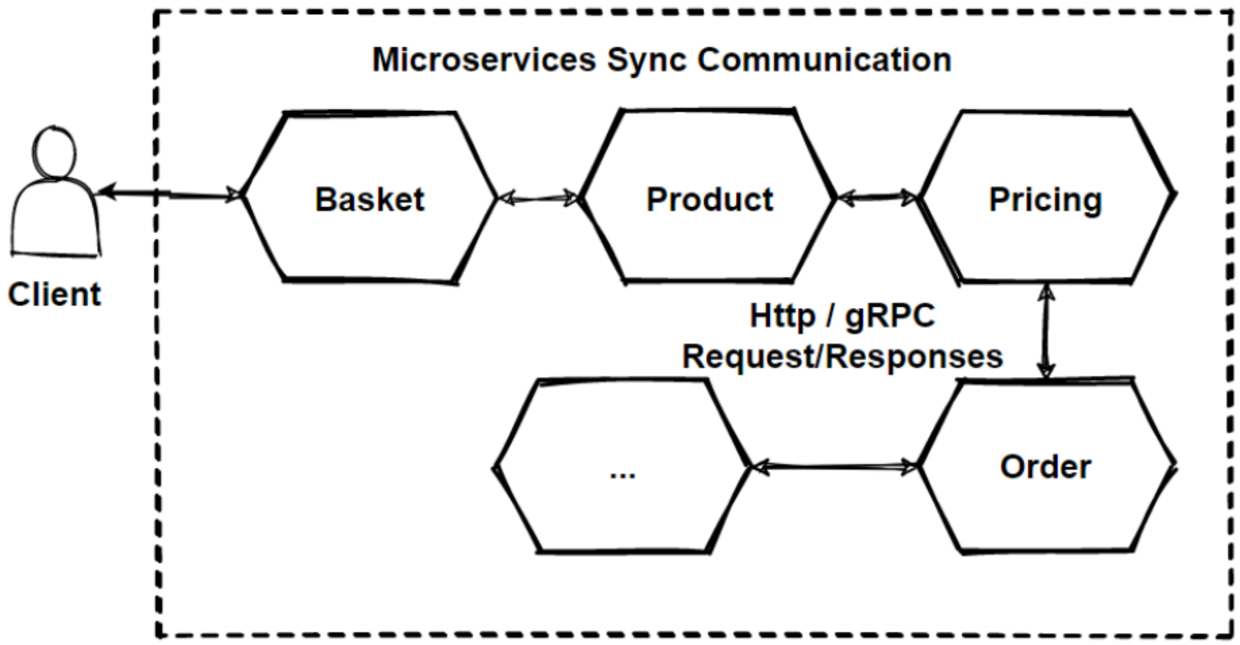
```
DaprPublish publish = new DaprPublish();
publish.publishEvent("pubsub", "topic-name", "Hello from Java!");
```

The Go Lang microservice subscribes to the topic and processes the message accordingly.

Technical Details and Implementation

Polyglot Microservices Communication

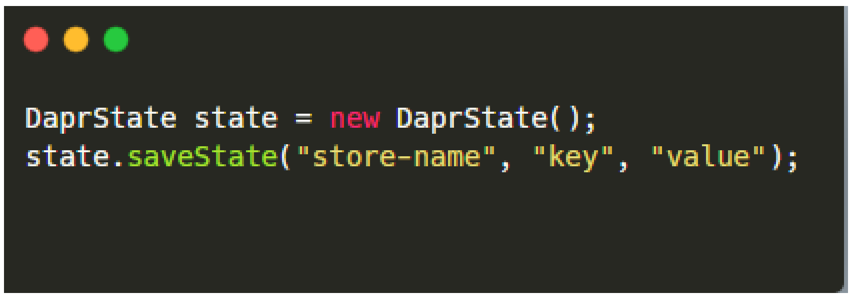
Dapr's language-agnostic nature simplifies communication between microservices.



It abstracts away the complexities of language-specific communication protocols, allowing developers to focus on business logic rather than inter-service communication details.

State Management

Dapr's state management feature allows microservices to maintain their state without tightly coupling to a specific storage solution. This is particularly useful when dealing with data persistence across different microservices.



```
DaprState state = new DaprState();
state.saveState("store-name", "key", "value");
```

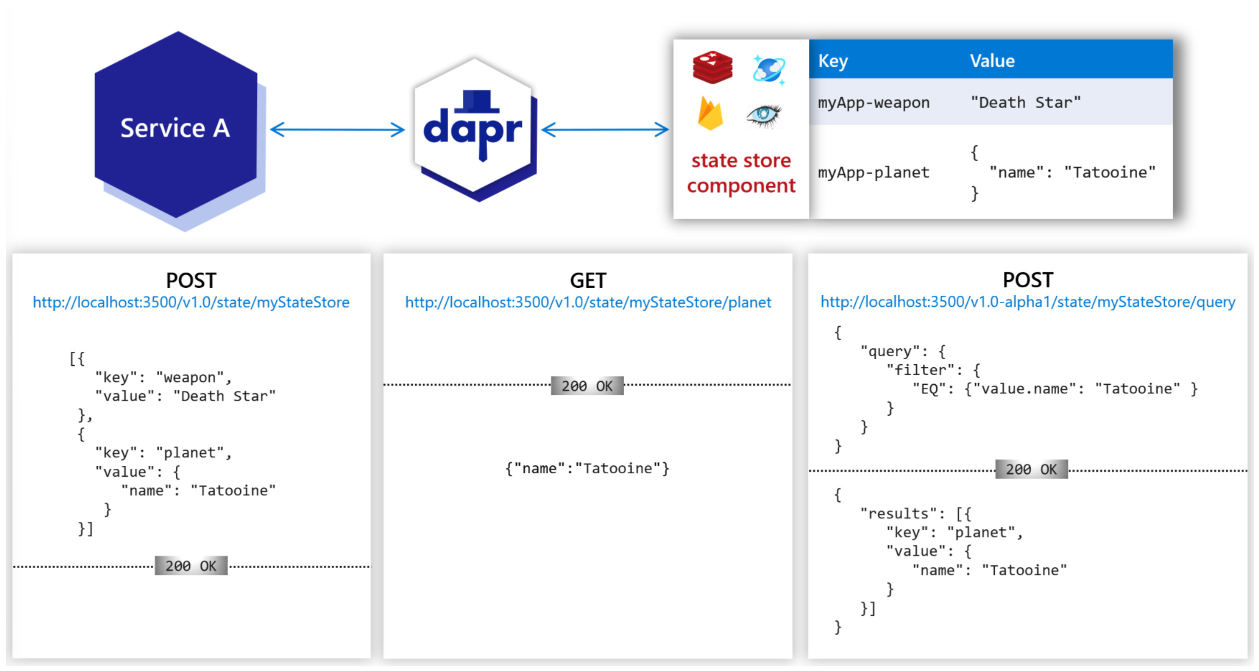
state.savestate("store-name", "key", "value");:

This line calls a method named savestate on the state object to store a piece of data in a Dapr state store.

"store-name": The name of the state store where the data should be saved. Dapr supports various state stores like Redis, Azure Cosmos DB, etc.

"key": A unique identifier used to retrieve the data later.

"value": The actual data to be stored, represented as a string in this case.



Key	Value
myApp-weapon	"Death Star"
myApp-planet	{ "name": "Tatooine" }

POST
http://localhost:3500/v1.0/state/myStateStore

```
[{
   "key": "weapon",
   "value": "Death Star"
 },
 {
   "key": "planet",
   "value": {
      "name": "Tatooine"
   }
 }]
```

200 OK

GET
http://localhost:3500/v1.0/state/myStateStore/planet

200 OK

```
{"name":"Tatooine"}
```

POST
http://localhost:3500/v1.0-alpha1/state/myStateStore/query

```
{
   "query": {
      "filter": {
         "EQ": {"value.name": "Tatooine" }
      }
   }
}
```

200 OK

```
{
   "results": [{
      "key": "planet",
      "value": {
         "name": "Tatooine"
      }
   }]
}
```

Various components connected to the state store, representing functionalities like:

Client Applications: Microservices interacting with the state store to save and retrieve data.

State Management APIs: Functions like savestate and getstate used for data operations.

Event Buses: Channels for propagating updates or triggers related to state changes.

External Systems: Integrations with databases, caches, or other external data sources.

Challenges in Implementing the Solution

While adopting polyglot microservices with Dapr offers numerous advantages, challenges may arise during implementation:

- Developers unfamiliar with Dapr may face a learning curve. Training and documentation play a crucial role in overcoming this challenge.

- Ensuring consistency in naming conventions, error handling, and logging across different programming languages can be challenging. Establishing coding guidelines can mitigate this issue.

- Tooling Support: Although Dapr supports multiple languages, some languages might have better tooling and support than others. Addressing these disparities may require additional effort.

Business Benefit

- With polyglot microservices, development teams can choose the best-suited language for their microservices, promoting flexibility and autonomy.

- Dapr simplifies the implementation of scalable and resilient microservices by providing building blocks for common distributed system challenges.

- Developers can focus on business logic rather than the intricacies of inter-service communication, resulting in faster development cycles and quicker time-to-market.

- Polyglot microservices with Dapr allow seamless integration with existing systems, including legacy applications, by providing a language-agnostic communication layer.

Conclusion:

In conclusion, adopting polyglot microservices with Dapr on Microsoft Azure opens up new possibilities for developers. The seamless integration of Java-Springboot, .NET, Go Lang, and Rust microservices promotes flexibility, scalability, and resilience in the ever-evolving world of distributed systems.

Incorporating **Microsoft Azure** as the underlying platform adds another layer of robustness and scalability to the solution, making it well-suited for enterprise-level applications. As technology continues to advance, embracing polyglot microservices with Dapr becomes not just a choice but a strategic advantage in building the next generation of scalable and resilient applications.

By Nixitha Boga
(nixitha67@gmail.com)