devonfw getting started guide

Updated at 2020-09-08 05:50:10 UTC

Table of Contents

Part I: Getting Started

1. Introduction

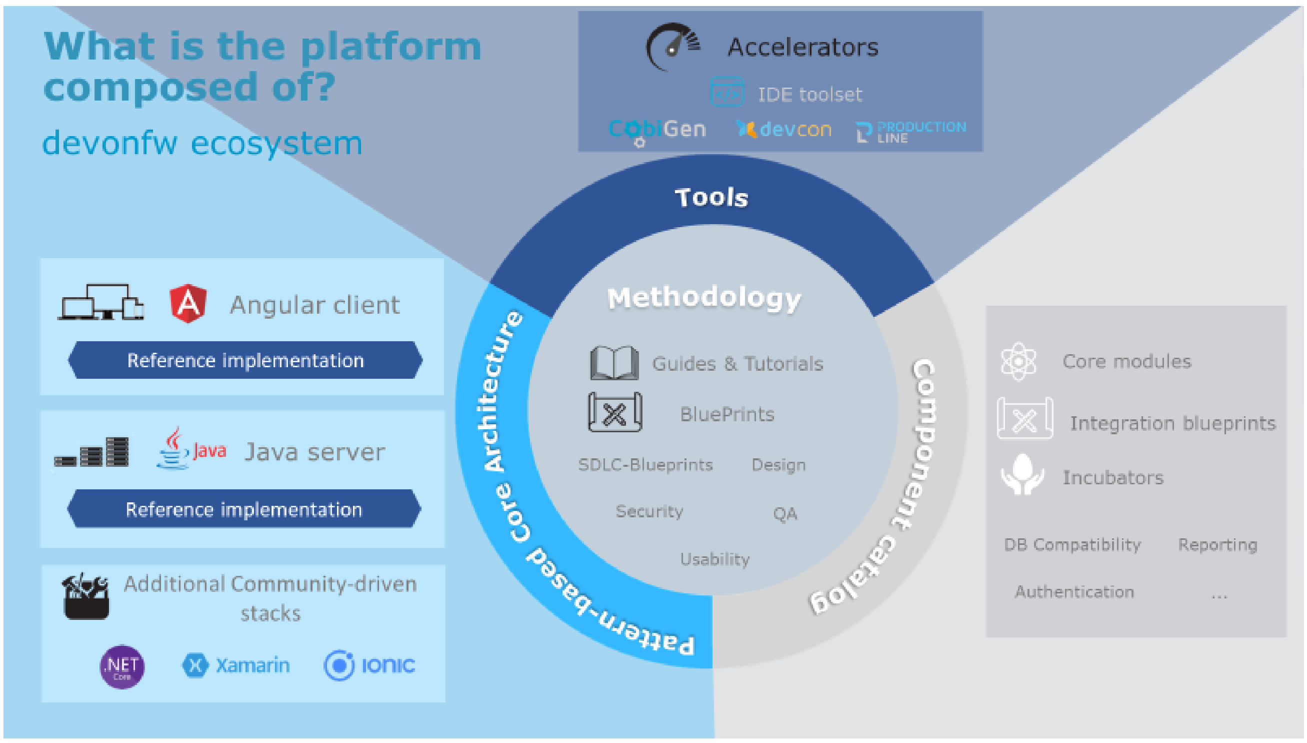
1.1. What is devonfw?

Welcome to the **devonfw** platform. This is a product of the CSD (Custom Solution Development) industrialization effort to establish a standardized platform for custom software development within Capgemini APPS2. This platform is aimed at engagements, in which clients don’t specify the use of a predefined technology stack. In these cases we can offer a proven alternative as a result of our experience as a group.

devonfw is a development platform aiming for the standardization of processes and the boosting of productivity. It provides an architecture blueprint for server and client applications, alongside a set of tools to deliver a fully functional, *out-of-the-box* development environment.

The devonfw *name* is a registered trademark of Capgemini, but the software and documentation included in *devonfw* are fully open source. Please refer to our OSS Compliance section for more information.

1.1.1. Building Blocks of the Platform



devonfw uses a state-of-the-art, open source, core reference architecture for the server (these days considered a commodity in the IT-industry) and on top of that an ever increasing number of high-value assets, which are developed by Capgemini.

1.1.2. The devonfw Technology Stack

devonfw is fully open source and consists of the following technology stacks:

Back-End Solutions

For server applications, *devonfw* includes the following solutions:

- devon4j: Server implementation based on Java, Spring and Spring Boot.
- devon4net: Server implementation based on .NET.
- devon4node: Server implementation based on NestJS.

Front-End solutions

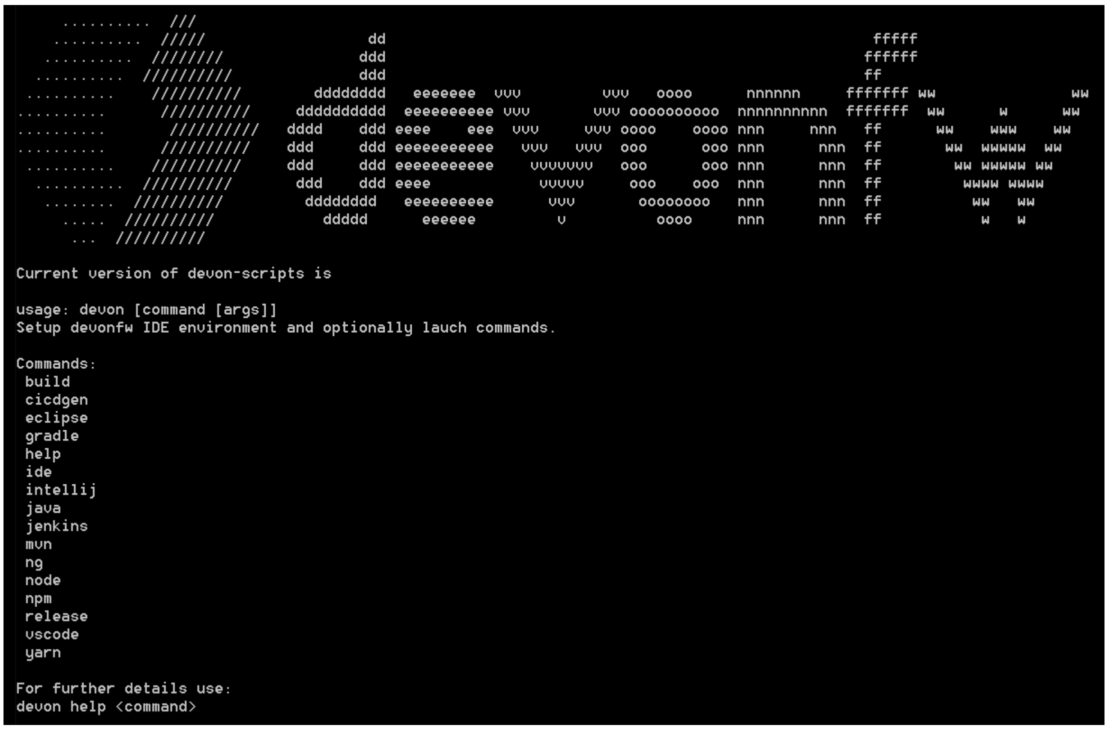
For client applications, *devonfw* includes two solutions based on *TypeScript*, *JavaScript*, *C#* and *.NET*:

- devon4ng: Frontend implementation based on Angular and a hybrid mobile implementation based on Ionic.
- devon4X: Mobile implementation based on Xamarin.

1.1.3. Custom Tools

devonfw-ide

The devonfw-ide is *not* one monolithic program that is installed with a traditional executable; rather it's a collection of scripts which are invoked via command line to automate several, repetetive development tasks. These scripts then interact with other tools, frameworks, and third-party IDEs to streamline the development workflow.



```
     ..........  ///
    ..........  /////                  dd                                                      fffff
   ..........  ////////               ddd                                                     ffffff
  ..........  //////////              ddd                                                     ff
 ..........    //////////        dddddddd   eeeeeee  vvv         vvv   oooo      nnnnnn     fffffff ww               ww
..........      //////////     dddddddddd  eeeeeeeeee vvv       vvv oooooooooo  nnnnnnnnn   fffffff  ww      w      ww
..........       //////////   dddd    ddd eeee    eee  vvv     vvv oooo    oooo nnn     nnn   ff      ww    www    ww
..........      //////////    ddd     ddd eeeeeeeeeee   vvv   vvv  ooo      ooo nnn      nnn  ff       ww  wwwww  ww
 ..........    //////////     ddd     ddd eeeeeeeeeee    vvvvvvv   ooo      ooo nnn      nnn  ff        ww wwwww ww
  ..........  //////////       ddd    ddd eeee            vvvvv     ooo    ooo  nnn      nnn  ff         wwww wwww
   ........  //////////         dddddddd   eeeeeeeeee      vvv       oooooooo   nnn      nnn  ff          ww   ww
     .....  //////////            ddddd      eeeeee         v          oooo     nnn      nnn  ff           w   w
      ...  //////////

Current version of devon-scripts is

usage: devon [command [args]]
Setup devonfw IDE environment and optionally lauch commands.

Commands:
 build
 cicdgen
 eclipse
 gradle
 help
 ide
 intellij
 java
 jenkins
 mvn
 ng
 node
 npm
 release
 vscode
 yarn

For further details use:
devon help <command>
```

The advantage of this approach is, that you can have as many instances of the *devonfw-ide* on your machine as you need — for different projects with different tools, tool versions and configurations. No need for a physical installation and no tweaking of your operating system required!

Instances of the devonfw-ide do not interfere with each other, nor with other installed software. The package size of the *devonfw-ide* is initally very small, the setup is simple, and the included software is portable.

IDEs

It supports the following IDEs:

- Eclipse
- Visual Studio Code
- IntelliJ IDEA

Platforms

It supports the following platforms:

- Java (see also devon4j)
- Node.js (see also devon4node)
- Angular (see also devon4ng)
- C# (see also devon4net)

Build-Systems

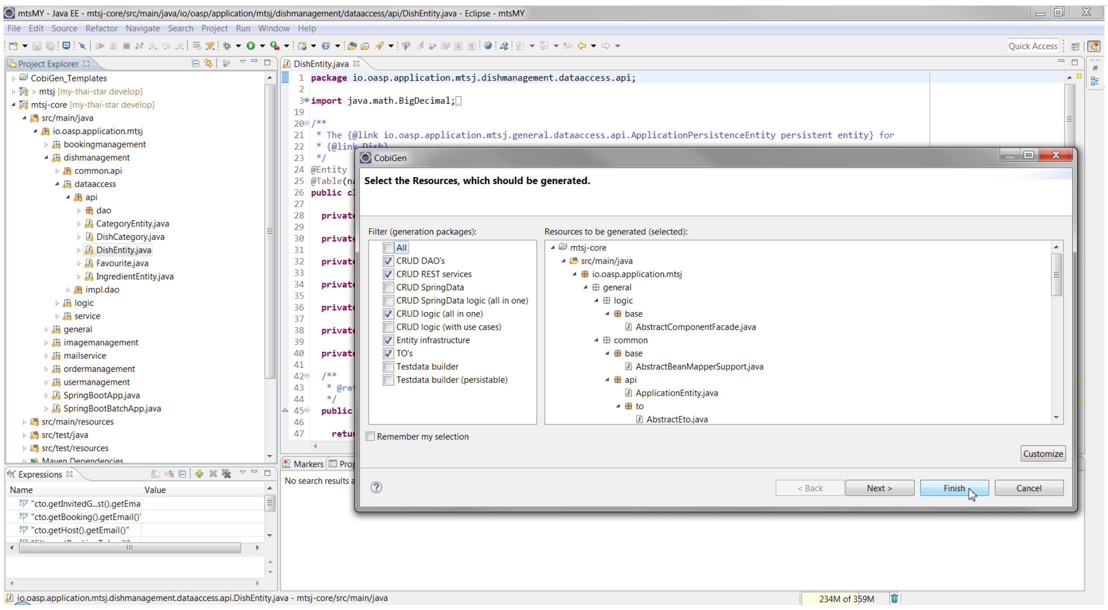
It supports the following build-systems:

- Maven
- NPM
- Gradle

Other IDEs, platforms, or tools can easily be integrated as commandlets.

CobiGen

CobiGen is a code generator included in the *devonfw-ide*, that allows users to generate the project structure and large parts of the application component code. This saves a lot of time, which is usually wasted on repetitive engineering tasks and/or writing boilerplate code.


CobiGen
Select the Resources, which should be generated.
Filter (generation packages):
[ ] All
[✓] CRUD DAO's
[✓] CRUD REST services
[ ] CRUD SpringData
[ ] CRUD SpringData logic (all in one)
[✓] CRUD logic (all in one)
[ ] CRUD logic (with use cases)
[✓] Entity infrastructure
[✓] TO's
[ ] Testdata builder
[ ] Testdata builder (persistable)
Resources to be generated (selected):
mtsj-core
src/main/java
io.oasp.application.mtsj
general
logic
base
AbstractComponentFacade.java
common
base
AbstractBeanMapperSupport.java
api
ApplicationEntity.java
to
AbstractEto.java
[ ] Remember my selection
Customize
< Back
Next >
Finish
Cancel

Following the same philosophy as the devonfw-ide, *CobiGen* bundles a new command line interface (CLI), that enables the generation of code using only a few commands. This approach also allows us to decouple CobiGen from *Eclipse* and use it alongside *VS Code* or *IntelliJ IDEA*.

1.2. Why should I use devonfw?

devonfw aims to provide a framework for the development of web applications based on the Java EE programming model. It uses the Spring framework as its Java EE default implementation.

1.2.1. Objectives

Standardization

We don’t want to keep reinventing the wheel for thousands of projects, for hundreds of customers, across dozens of countries. For this reason, we aim to rationalize, harmonize and standardize the development assets for software projects and industrialize the software development process.

Industrialization of Innovative Technologies & “Digital”

devonfw’s goal is to standardize & industrialize. But this applies not only to large volume, “traditional” custom software development projects. devonfw also aims to offer a standardized platform which contains a range of state-of-the-art methodologies and technology stacks. devonfw supports agile development by small teams utilizing the latest technologies for projects related to Mobile, IoT and the Cloud.

Deliver & Improve Business Value

Efficiency

- Up to 20% reduction in time to market, with faster delivery due to automation and reuse.
- Up to 25% less implementation efforts due to code generation and reuse.
- Flat pyramid and rightshore, ready for junior developers.

Quality

- State-of-the-art architecture and design.
- Lower cost on maintenance and warranty.
- Technical debt reduction by reuse.
- Risk reduction due to continuous improvement of individual assets.
- Standardized, automated quality checks.

Agility

- Focus on business functionality, not on technicalities.
- Shorter release cycles.
- DevOps by design — Infrastructure as Code.
- Continuous Delivery pipeline.
- On- and off-premise flexibility.
- PoCs and prototypes in days not months.

1.2.2. Features

Everything in a Single ZIP

The devonfw distributions is packaged in a ZIP file that includes all the custom tools, software and configurations.

Having all the dependencies self-contained in the distribution’s ZIP file, users don’t need to install or configure anything. Just extracting the ZIP content is enough to have a fully functional *devonfw*.

devonfw — The Package

The devonfw platform provides:

- Implementation blueprints for a modern cloud-ready server and a choice on JS-Client technologies (either open source Angular or a very rich and impressive solution based on commercial Sencha UI).
- Quality documentation and step-by-step quick start guides.
- Highly integrated and packaged development environment based around Eclipse and Jenkins. You will be ready to start implementing your first customer-specific use case in 2h time.
- Iterative eclipse-based code-generator that understands "Java" and works on higher architectural concepts than Java-classes.
- An example application as a reference implementation.
- Support through a large community + industrialization services (Standard Platform as a Service) available in the iProd service catalog.

1.3. devonfw-ide Download and Setup

Please refer to our devonfw-ide Setup section.

2. Guides

Our goal is to provide a smooth starting experience to all users of *devonfw*, no matter how experienced they are or what their stakeholder role is. To achieve this, we provide a list of recommended guides here:

For Students and Junior Engineers:

- Explore the devonfw sample application.
- Build your first application with **devon4j/devon4ng**.
- Create an application with **devon4node**.
- Create an application with **devon4net**.

For Senior Engineers and Architects:

- Explore the devonfw sample application.
- Explore the guide to implement ISTIO features.

For Team Leaders and Product Ambassadors:

- License management for OSS projects.

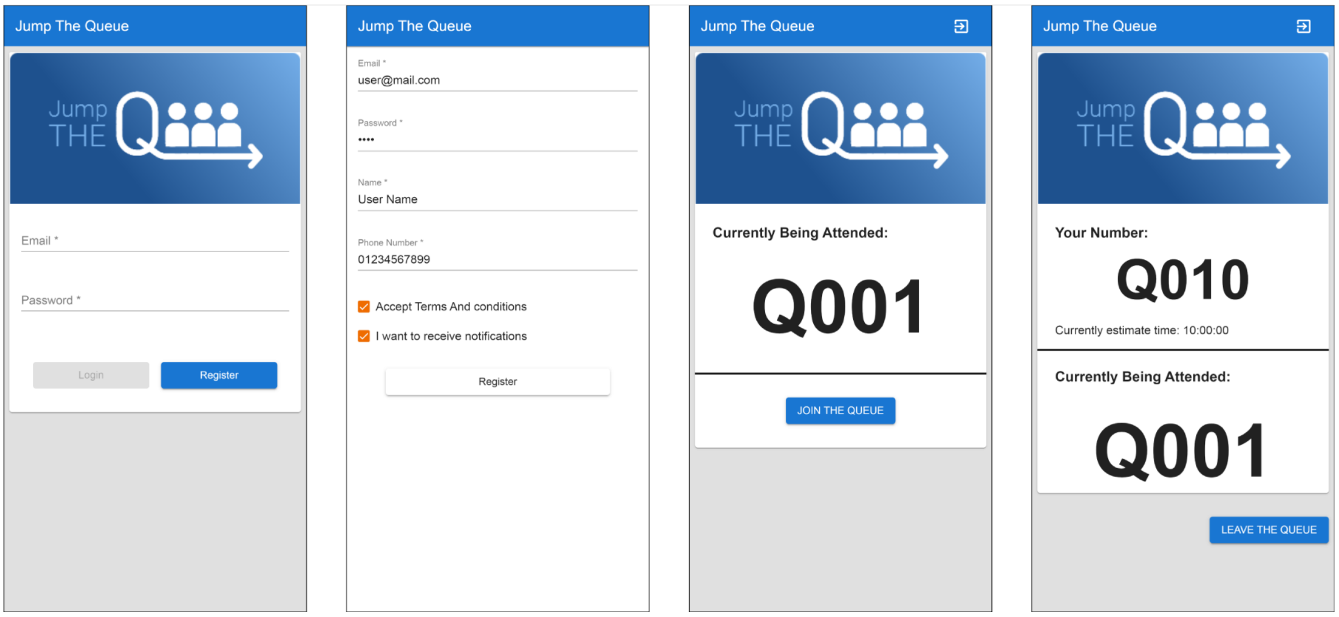
2.1. Build Your First devonfw Application

JumpTheQueue is a small application based on the devonfw framework, which you can create yourself by following our simple step-by-step tutorial. By doing so, you will learn about the app development workflow and gain insight into the design of a professional business information system. Please visit the JumpTheQueue wiki and start working trough the tutorial **HERE**.

The tutorial assumes you have successfully set up the devonfw-ide previously.

You can also clone the project and explore the finished source code via:

```
git clone https://github.com/devonfw/jump-the-queue.git
```



2.2. Explore Our devonfw Sample Application

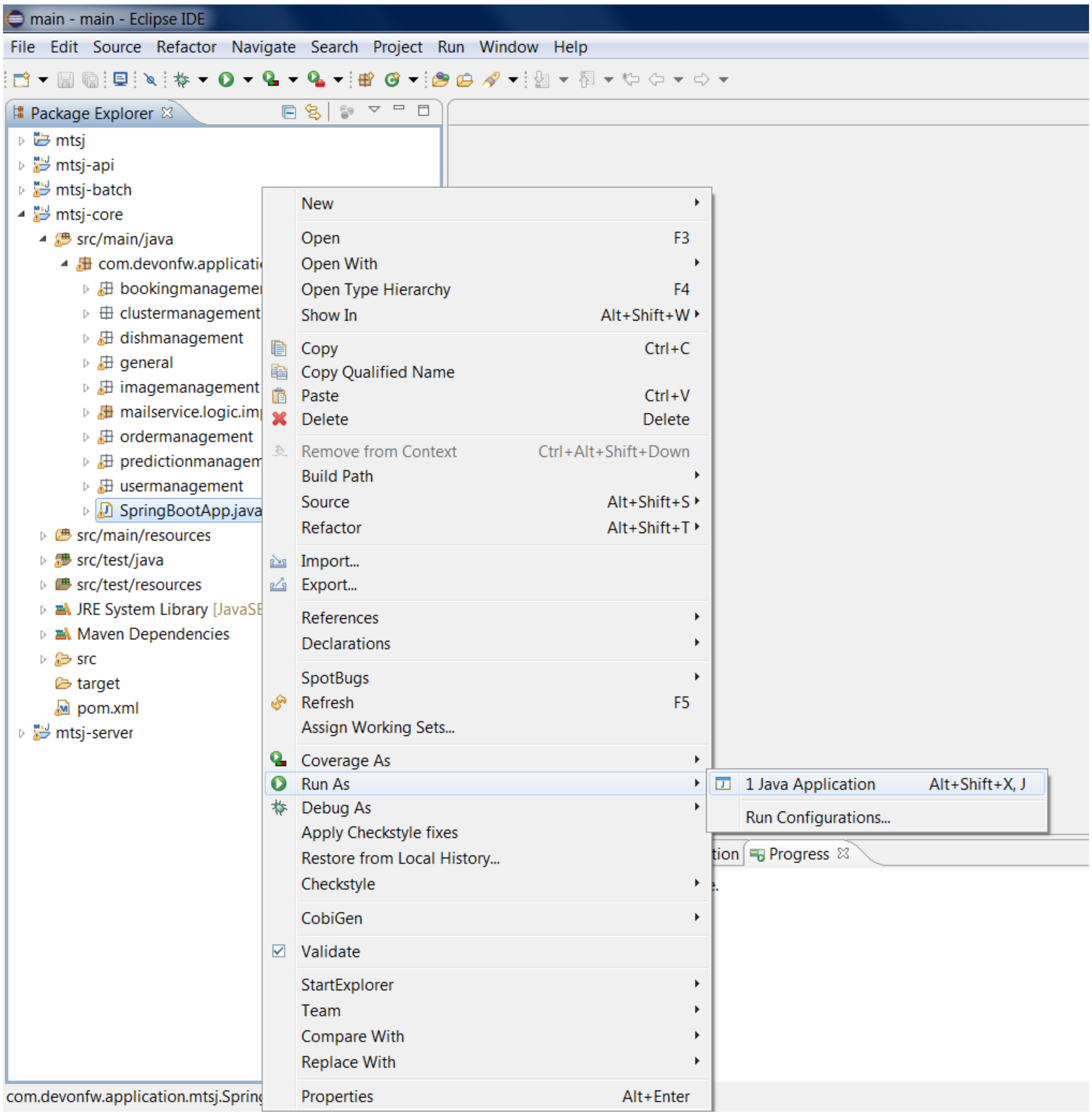
MyThaiStar is a complex sample app, that demonstrates the full capabilities of our framework. On this page we will describe how to download and launch the app on your system, so you can test the various functionalities it offers and explore its code.

We assume you have successfully set up the devonfw-ide previously.

1. In the root directory of a *devonfw-ide* directory, right click and select "**Open Devon CMD shell here**" from the Windows Explorer context menu. Then navigate to the main workspace and checkout the MyThaiStar Git repository like this:

```
cd workspaces/main
git clone https://github.com/devonfw/my-thai-star.git
```

2. Perform: `cd my-thai-star`
3. Execute: `devon eclipse ws-up`
4. Execute: `devon eclipse create-script`
5. Go to the root folder of the distribution and run `eclipse-main.bat`
6. In Eclipse navigate to `File > Import > Maven > Existing Maven Projects`, then import the cloned project from your workspace by clicking the "Browse" button and selecting `/workspaces/my-thai-star/java/mtsj/`.
7. Run the backend by right-clicking `SpringBootApp.java` and selecting `Run as > Java Application` in the context menu. The backend will start up and create log entries in the Eclipse Console tab.



8. Return to your command shell and perform: `cd angular`
9. Execute: `npm install`
10. Execute: `ng serve`
11. Once started, the frontend will be available at localhost:4200/restaurant. Login with the username and password `waiter` and take a look at the various functionalities provided by MyThaiStar.

You should now take a look at both the front- and backend code and familiarize yourself with its structure and concepts, since most devonfw projects follow this exemplary implementation. Please visit the *architecture overview* pages of devon4ng and devon4j to learn more about the internal workings of front- and backend.

2.3. ISTIO Service Mesh Implementation Guide

2.3.1. Introduction

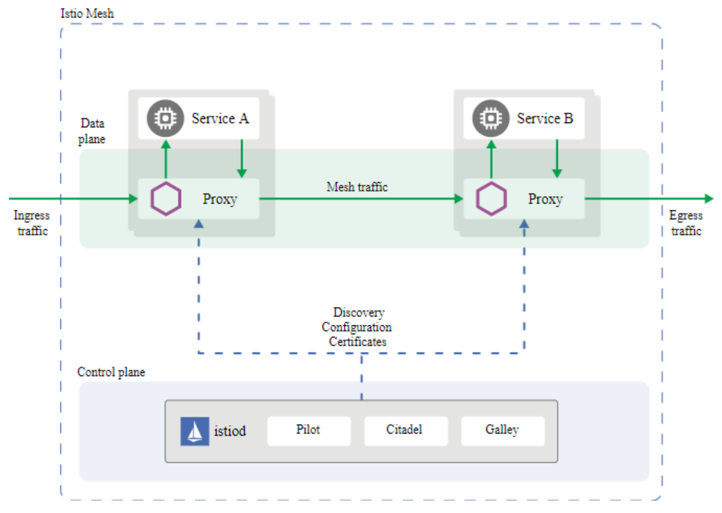
A service mesh separating applications from network functions like resilience, fault tolerance, etc,.

A service mesh addresses the below functions without changing the application code.

- Test the new versions of services without impacting the users.
- Scale the services.
- Find the services with the help of service registry.
- Test against failures.
- Secure service-to-service communication.
- Route traffic to a specific way.
- Circuit breaking and fault injection.
- Monitor the services and collect matrices.
- Tracing.

ISTIO service mesh is an open environment for Connecting, Securing, Monitoring services across the environments.

2.3.2. ISTIO Architecture

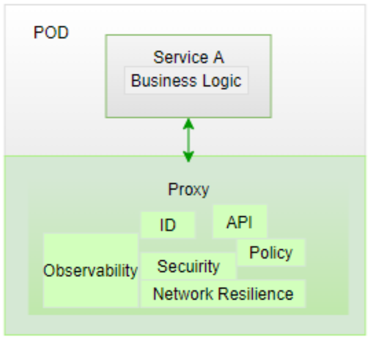


Istio Architecture

ISTIO is split into data plane and control plane.

Data Plane

The data plane is a set of intelligent proxies (Envoy) deployed as sidecars that mediate and control all network communication among microservices.



Control Plane

The control plane is managing and configuring proxies to route traffic and enforcing policies.

- Pilot manages all the proxies and responsible for routing
- Mixer collects telemetry and policy check
- Citadel does Certificate management (TLS certs to Envoys)

2.3.3. ISTIO installation

Download ISTIO from releases

```
istioctl install --set profile=demo
```

Here used demo profile, there are other profiles for production.

Verify installation:

```
kubectl get all -n istio-system
```

Inject sidecar container automatically by issuing the below command.

```
kubectl label namespace default istio-injection=enabled
```

Verify:

```
kubectl get namespace -L istio-injection
```

2.3.4. Traffic Management

ISITO’s traffic management model relies on the Envoy proxies which deployed as sidecars to services.

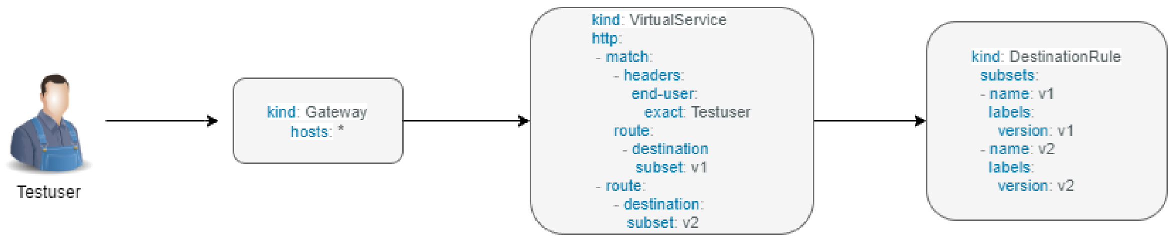
Below are the traffic management API resources

- Virtual Services
- Destination Rules
- Gateways
- Service Entries
- Sidecars

A virtual service, higher level abstraction of Kubernetes Service, lets you configure how requests are routed to a service within an Istio service mesh. Your mesh may have multiple virtual services or none. Virtual service consists of routing rules that are evaluated in order.

Dark Launch

The following virtual service routes requests to different versions of a service depending on whether the request comes from a testuser. If the testuser calls then version v1 will be used, and for others version v2.



```
kind: Gateway
hosts: *
```

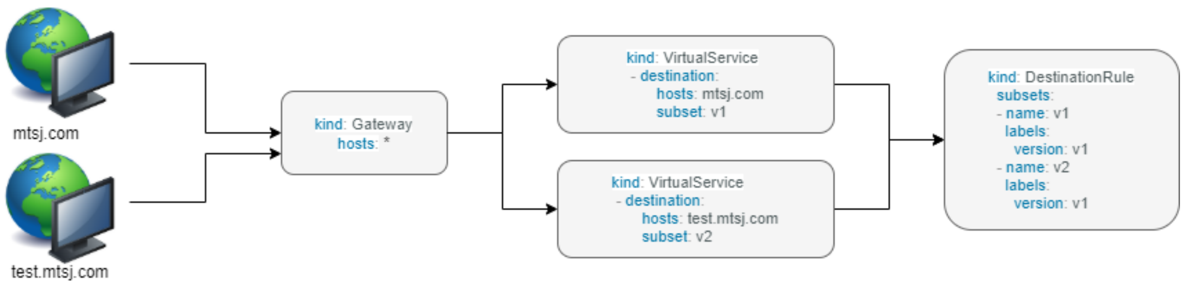
```
kind: VirtualService
http:
- match:
  - headers:
      end-user:
        exact: Testuser
    route:
    - destination
      subset: v1
- route:
  - destination:
    subset: v2
```

```
kind: DestinationRule
subsets:
- name: v1
  labels:
    version: v1
- name: v2
  labels:
    version: v2
```

Blue/Green deployment

In blue/green deployment two versions of the application running. Both versions are live on different domain names, in this example it is mtsj.com and test.mtsj.com.

1. Define 2 virtual services for mtsj v1 and v2 versions.
2. Define DestinationRule and configure the subsets for v1 and v2.



```
kind: Gateway
hosts: *
```

```
kind: VirtualService
- destination:
    hosts: mtsj.com
    subset: v1
```

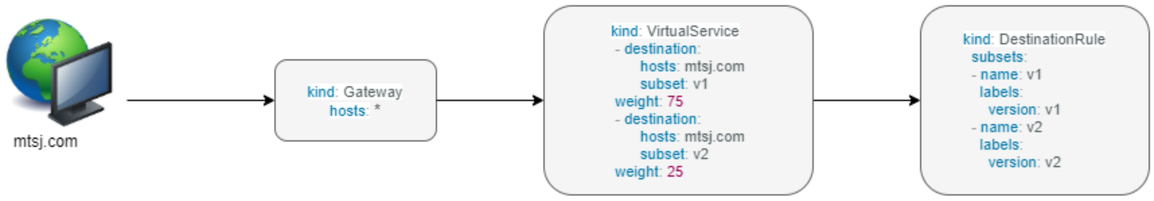
```
kind: VirtualService
- destination:
    hosts: test.mtsj.com
    subset: v2
```

```
kind: DestinationRule
subsets:
- name: v1
  labels:
    version: v1
- name: v2
  labels:
    version: v1
```

When end user browses *mtsj.com*, the gateway call goes to subset v1 of the virtual service and redirects to destination version v1, and for *test.mtsj.com* to version v2.

Canary Deployment (Traffic Splitting)

In canary deployment old and new versions of the application alive. ISTIO can be configured, how much percentage of traffic can go to each version.



```
kind: Gateway
hosts: *
```

```
kind: VirtualService
- destination:
    hosts: mtsj.com
    subset: v1
weight: 75
- destination:
    hosts: mtsj.com
    subset: v2
weight: 25
```

```
kind: DestinationRule
subsets:
- name: v1
  labels:
    version: v1
- name: v2
  labels:
    version: v2
```

Here, the traffic is divided 75% to the version V1, and 25% to the version V2, as we gain confidence the percentage can be increased the latest version and gradually the traffic to the old version can be reduced and removed.

MyThaiStar Implementation

In this example dish will have two versions and the traffic will be routed alternately using the ISTIO configuration.

Find all configuration files in istio/trafficmanagement/canary directory under mythaistarmicroservices example.

1. MyThaiStar defines below
   a. Service
   b. Service Account
   c. Deployment

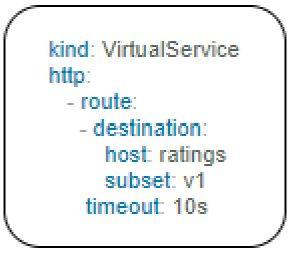
The above configurations are defined in a single yaml file for all the different services like angular, dish, image etc.

1. dish-v2: Dish Version 2 can be kept separately in different yaml file.
2. mts-gateway defines the ingress gateway which routes the outbound request to each service.
3. destination-rule-all defines the subsets here for later traffic routing
4. dish-50-50: traffic routing for different versions of dishmanagement.

Network Resilience

Timeout

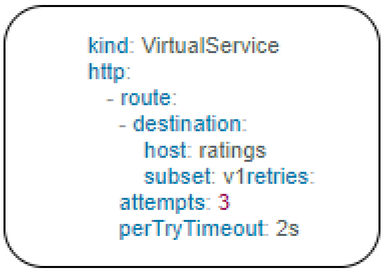
Istio lets you adjust the timeouts using virtual services. The default timeout is 15 seconds.



```
kind: VirtualService
http:
  - route:
    - destination:
        host: ratings
        subset: v1
      timeout: 10s
```

Retry

A retry setting specifies the maximum number of times an Envoy proxy attempts to connect to a service if the initial call fails.



```
kind: VirtualService
http:
  - route:
    - destination:
        host: ratings
        subset: v1retries:
    attempts: 3
    perTryTimeout: 2s
```

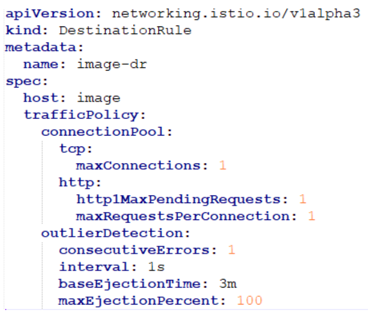
Retries can also be configured on Gateway Error, Connection failure, Connection Refused or any 5xx error from the application.

retryOn: gateway-error,connect-failure,refused-stream,5xx

Circuit Breakers

By defining the destination rule, set limits for calls to individual hosts within a service, such as the number of concurrent connections or how many times calls to this host have failed once the limit reached.

- Outlier Detection is an ISTIO Resiliency strategy to detect unusual host behaviour and evict the unhealthy hosts from the set of load balanced healthy hosts inside a cluster.
- If a request is sent to a service instance and it fails (returns a 50X error code), then ISTIO ejects the instance from the load balanced pool for a specified duration.



```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: image-dr
spec:
  host: image
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 1
      http:
        http1MaxPendingRequests: 1
        maxRequestsPerConnection: 1
    outlierDetection:
      consecutiveErrors: 1
      interval: 1s
      baseEjectionTime: 3m
      maxEjectionPercent: 100
```

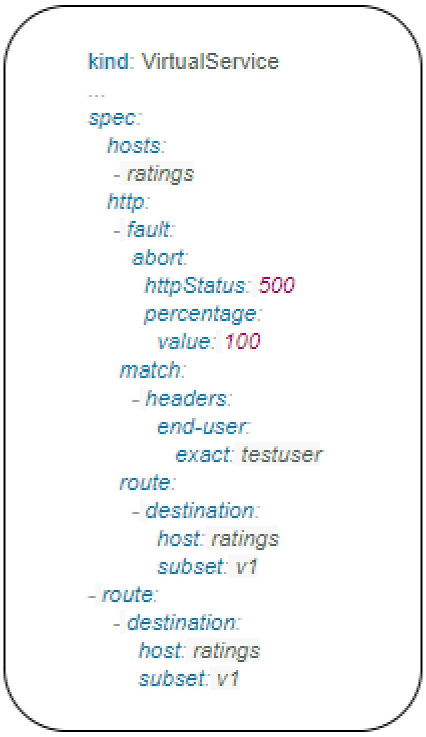
Fault Injection

Two types of faults can be generated using ISTIO. This is useful for the testing.

Delays: timing failures.

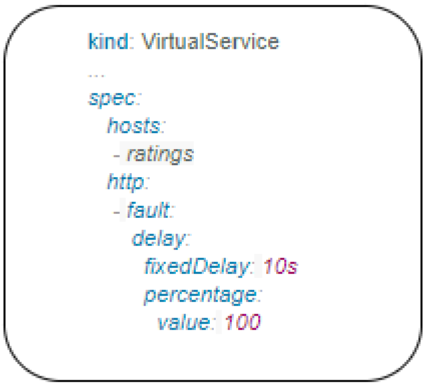
Aborts: crash failures.

Below example is a crash failure Virtual Service. The below example configured to receive http status 500 error for the testuser. The application works fine for all other users.



```
kind: VirtualService
...
spec:
  hosts:
   - ratings
  http:
   - fault:
      abort:
        httpStatus: 500
        percentage:
          value: 100
    match:
     - headers:
         end-user:
           exact: testuser
    route:
     - destination:
         host: ratings
         subset: v1
- route:
   - destination:
       host: ratings
       subset: v1
```

The below virtual service configured to wait 10s for all requests.



```
kind: VirtualService
...
spec:
  hosts:
   - ratings
  http:
   - fault:
      delay:
        fixedDelay: 10s
        percentage:
          value: 100
```

2.3.5. Security

ISTIO provides security solution has the below functions.

- Traffic encryption
- Mutual TLS and fine-grained access policies.
- Auditing tools

Authentication

ISTIO provides two types of authentication.

- Peer authentication, secures service to service authentication
- Request authentication is end user authentication to verify credential attached to the request.

Mutual TLS Authentication

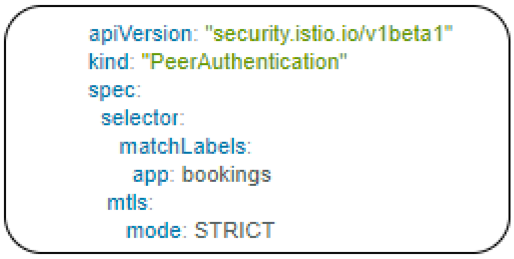
By default, the TLS protocol only proves the identity of the server to the client. Mutual TLS authentication ensures that traffic has been traffic is secure and trusted in both the directions between the client and server.

All traffic between services with proxies uses mutual TLS by default.

Peer Authentication

Peer authentication has Permissive, Strict and Disabled mode. With permissive mode, a service accepts both plain text and mutual TLS traffic. Permissive mode is good at the time of onboarding and should switch to Strict later.

The authentication policy can be applied to mesh-wide, namespace wide or workload specific using the selector field.



```
apiVersion: "security.istio.io/v1beta1"
kind: "PeerAuthentication"
spec:
  selector:
    matchLabels:
      app: bookings
   mtls:
     mode: STRICT
```

Here the policy applied to the workload bookings.

Check the default mesh policy:

`kubectl describe meshpolicy default`

Request authentication

Request authentication policies specify the values needed to validate JWT tokens.

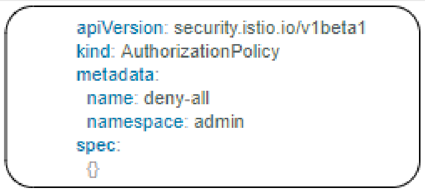
Authentication	Applies to	Uses	Identity
Peer authentication	Service to service	mTLS	source.principal
Request authentication	End User authentication	JWT	request.auth.principal

Authorization

Apply an authorization policy to the workload/namespace/mesh to enforce the access control. Supports ALLOW and DENY actions.

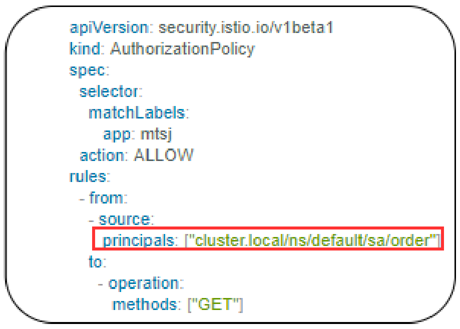
Deny All

Below example authorization policy without any rules denies access to all workloads in admin namespace.



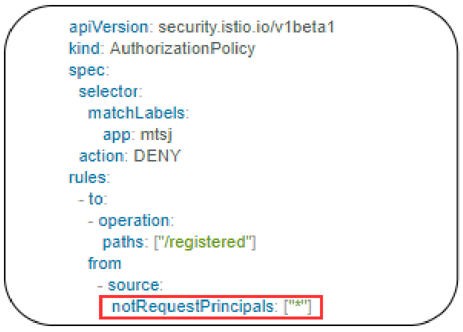
```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: admin
spec:
  {}
```

Example below allowing the GET methods from order service.



```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
spec:
  selector:
    matchLabels:
       app: mtsj
  action: ALLOW
rules:
  - from:
    - source:
       principals: ["cluster.local/ns/default/sa/order"]
    to:
      - operation:
         methods: ["GET"]
```

Example below denies the request to the /registered path for requests without request principals.



```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
spec:
  selector:
    matchLabels:
       app: mtsj
  action: DENY
rules:
  - to:
    - operation:
       paths: ["/registered"]
    from
      - source:
        notRequestPrincipals: ["*"]
```

2.3.6. Observability

ISTIO generates

- Metrics - for monitor latency, traffic, errors and saturation.
- Distributed Traces to identify call flows and service dependencies
- Access Logs enables audit service behaviour to the individual service level.

Grafana dashboard

Grafana and Prometheus are preconfigured addons on ISTIO. To enable, choose the configuration profile which has Prometheus and Grafana enabled. Eg: Demo profile

Verify Prometheus and Grafana running in the cluster.

```
kubectl get pods -n istio-system
```

Kiali dashboard

The Kiali dashboard helps you understand the structure of your service mesh by displaying the topology. The demo profile enables Kiali dashboard also.

Access the Kiali dashboard. The default user name is admin and default password is admin.

```
istioctl dashboard kiali
```

2.3.7. Minikube Troubleshooting Tips

This documentation provides the troubleshooting tips while working with minikube in a local machine.

1. Always start minikube with a minimum of 4GB of memory or more if available. Using command `minikube start --memory=4096`
2. If minikube is not starting or throwing any error even after multiple attempts. Try the below tips:
   a. Delete the minikube in your local machine using `minikube delete` and do a fresh minikube start.
   b. In any case, if minikube is not starting even after the above step, go to .minikube folder under the users directory and delete it manually. Now try starting minikube.
3. Set docker environment in minikube using `minikube docker-env`. Now all the docker commands that are run will be on the docker inside minikube. So building your application after executing the above command will have the application docker images available to minikube.
   a. To exit minikube docker environment use `minikube docker-env -u`
4. In any case, if you face any error related to docker image such as `Failed to pull image`, or `image not found` errors we will have to manually push the application docker image to minikube docker cache using the below commands.
5. For better results - stop minikube using `minikube stop` command.
6. Execute the command `minikube cache add imageName/tagName`.
7. Now start the minikube. To verify if the docker image has been added to minikube docker execute `minikube ssh docker images`.
8. To remove any docker image from minikube docker stop any containers running that docker image and then execute `minikube cache delete imageName/tagName`.
9. To reload any docker image to minikube docker environment, execute `minikube cache reload`.
10. In any case, if the docker images are not getting removed from minikube docker environment then navigate to .minikube/cache/images and then delete the particular image.

Execute the below command to make the Grafana available.

```
kubectl -n istio-system port-forward $(kubectl -n istio-system get pod -l app=grafana -o
jsonpath='\{.items[0].metadata.name}') 3000:3000 &
```

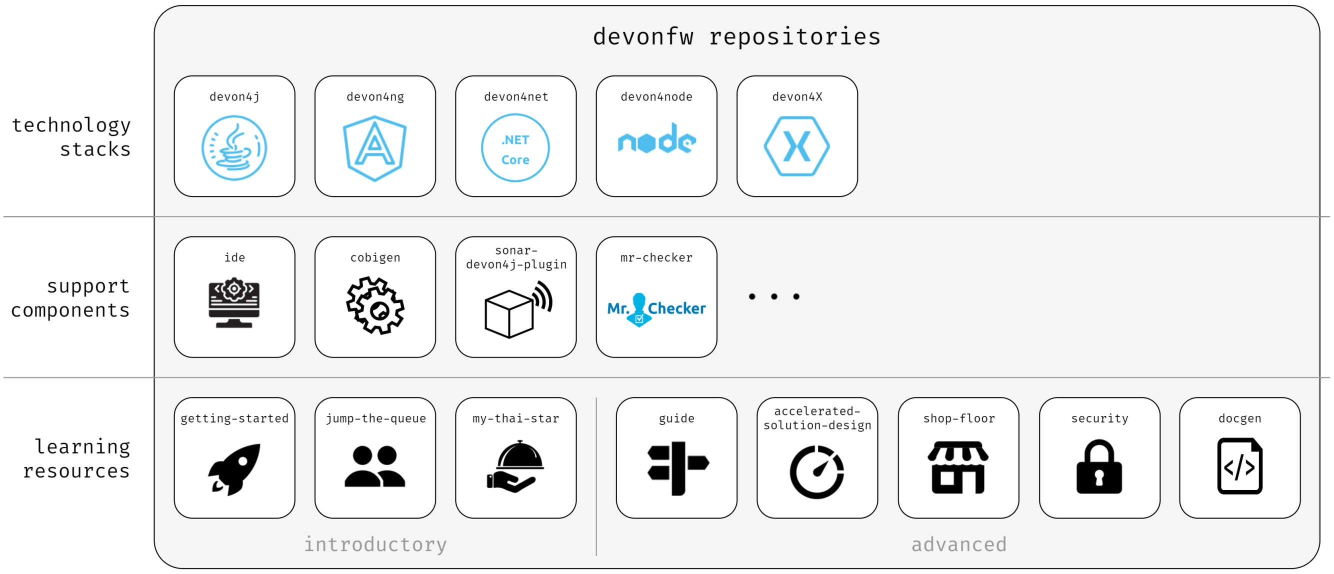
Use the below URLs to view the dashboard in local machine.

http://localhost:3000/dashboard/db/istio-mesh-dashboard

3. Further Information

3.1. Repository Overview

The GitHub repositories within the devonfw organization contain the source code and documentation for official devonfw projects.



An overview of the devonfw organization repositories.

The most relevant repositories here are the individual devonfw technology stacks:

- devon4j (for Java)
- devon4ng (for Angular)
- devon4net (for .NET)
- devon4node (for Node.js)
- devon4x (for Xamarin)

Our framework also delivers a number of tools and plug-ins that aim to accelerate and streamline the development process, for example:

- devonfw-ide (for environment/workspace setup)
- CobiGen (for automated code generation)
- devon4j-sonar-plugin (for architecture validation)
- MrChecker (for E2E test automation)

We also provide educational material and reference implementations to aid new users and drive the adoption of our framework, for example:

- JumpTheQueue (simple, step-by-step app creation tutorial)
- MyThaiStar (well documented, complex reference application)
- devonfw-shop-floor (tools and tutorials for CI/CD efforts)

- accelerated-solution-design (work-methodology for faster app design)
- and many more …

Projects in early development and prototypes are located in the devonfw forge repository. They usually remain there until they are ready for broader release or use in production.

3.2. Links to our Community

We strive to foster an active, diverse and dynamic community around devonfw and are relying on modern collaboration tools to do so. Please note that some resources listed here might only be accessible to members or partners of Capgemini.

3.2.1. Microsoft Teams

The devonfw public channel is accessible to everyone who has a Microsoft Teams account. You can find the latest discussions on ongoing development topics here, as well as new commits and pull requests to our repos.

Join us to stay in the loop, and feel free to post your questions regarding devonfw here.

devonfw Public Channel

3.2.2. Yammer

Our corporate Yammer channel is accessible to Capgemini employees and members. If you are looking for information or feedback on current and planned projects regarding devonfw, we reccomend you ask around here first.

devonfw Corporate Yammer

3.2.3. E-Mail

You can reach our dedicated iCSD Support Team via e-mail at:

icsddevonfwsupport.apps2@capgemini.com

3.3. Contributing

Please refer to our Contributing section.

3.4. Code of Conduct

Please refer to our Code of Conduct section.