SoS Automation
The Eclipse Arrowhead Framework

CPS Summer School 2024

Dr. Cristina Paniagua
Luleå University of Technology

ARROWHEAD
fPVN

Outline

- Theoretical background
- Hands-on tutorial
- Deploy your own system
- Inter-cloud communication
- Q&A – time to work!

Before start…

It needs to be ready for the hands-on part!

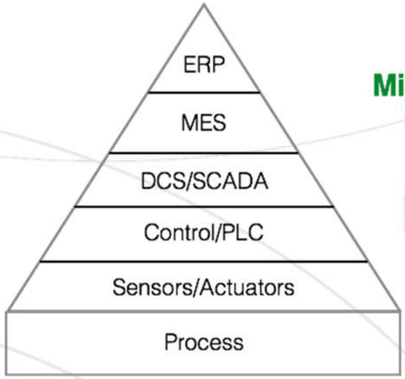
Clone repositories:

- https://github.com/eclipse-arrowhead/core-java-spring
- https://github.com/arrowhead-f/sos-examples-spring/tree/master

Theoretical Background

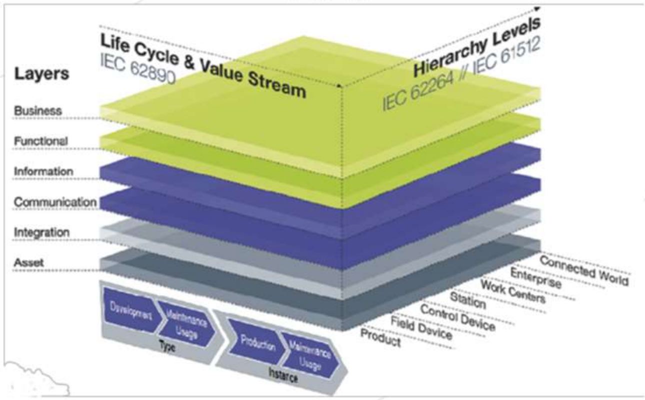
The automation technology transition

ISA95

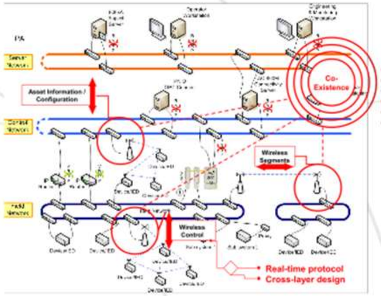


Migration path

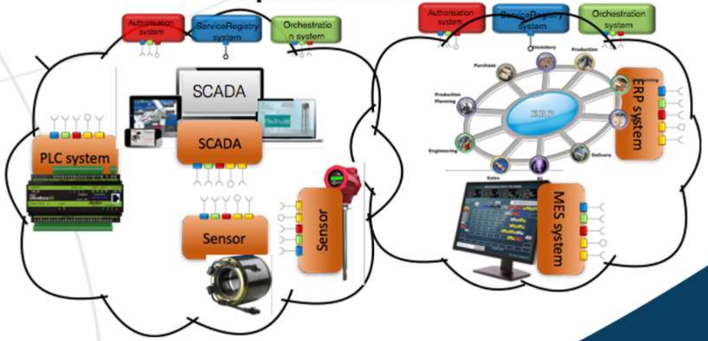
RAMI4.0



Hierarchical system implementation



Local automation cloud implementation



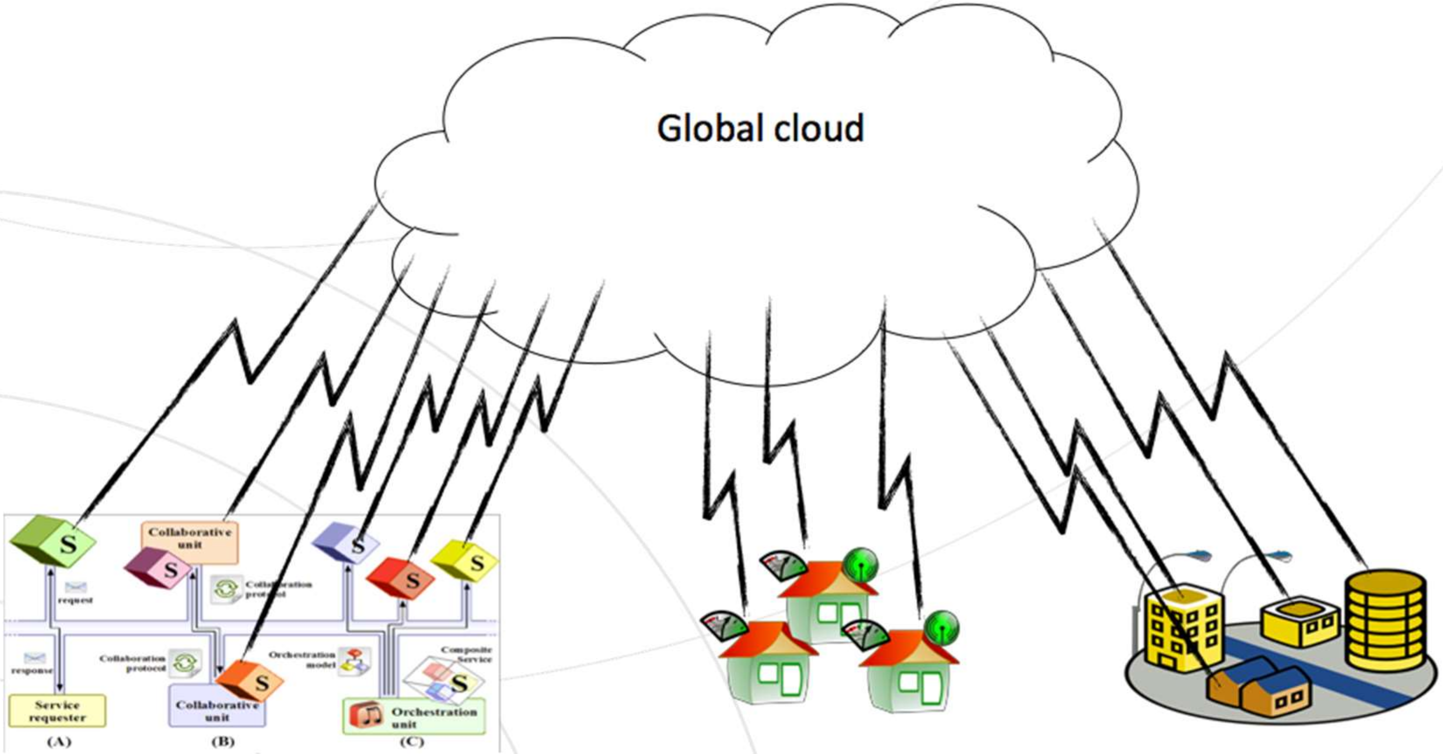
www.arrowhead.eu

ARROWHEAD
fPVN

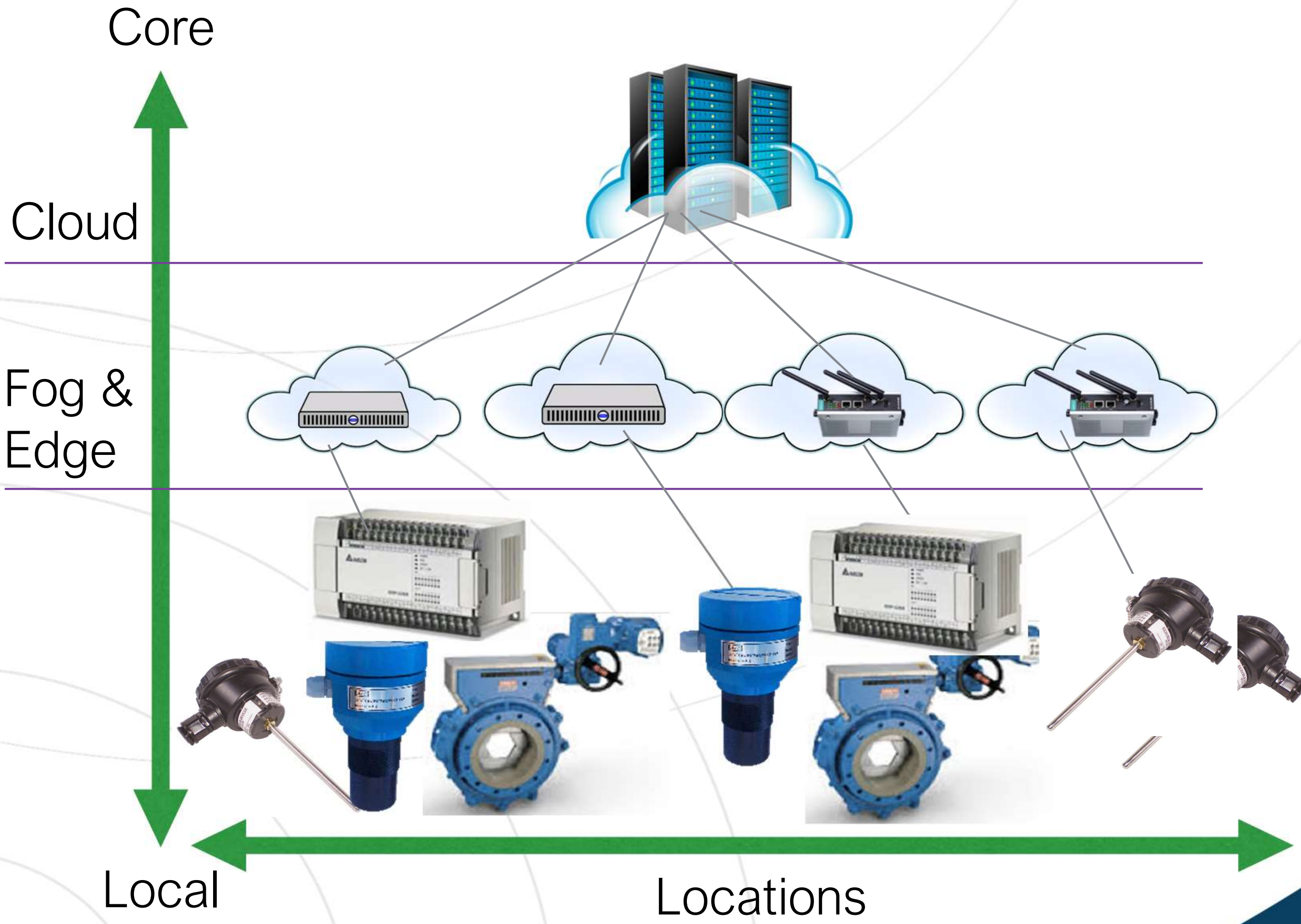
Digitalization and Automation System Requirements

- Real-time performance
- System safety
- Internet of Things (IoT), interoperability
- IT security and associated system safety
- Engineering simplicity to reduce, design- and run-time change, costs
- Scalability to very large system of systems (> 10^6 IoT's)
- System evolvability over time and technology generations

The global cloud approach



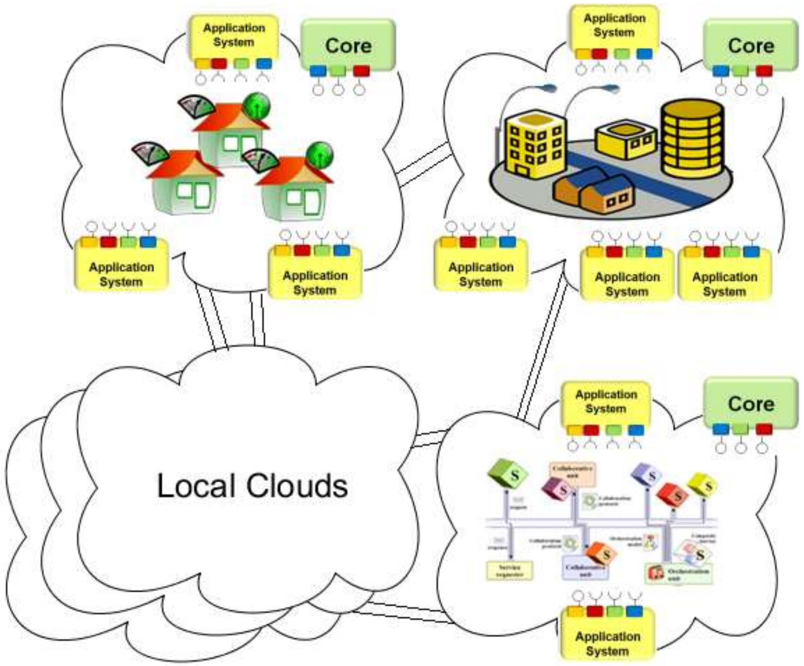
A Survey of Commercial Frameworks for the Internet of Things. Hasan Derhamy, Jens Eliasson, Jerker Delsing, and Peter Priller, SOCNE workshop at ETFA 2015, Luxemburg

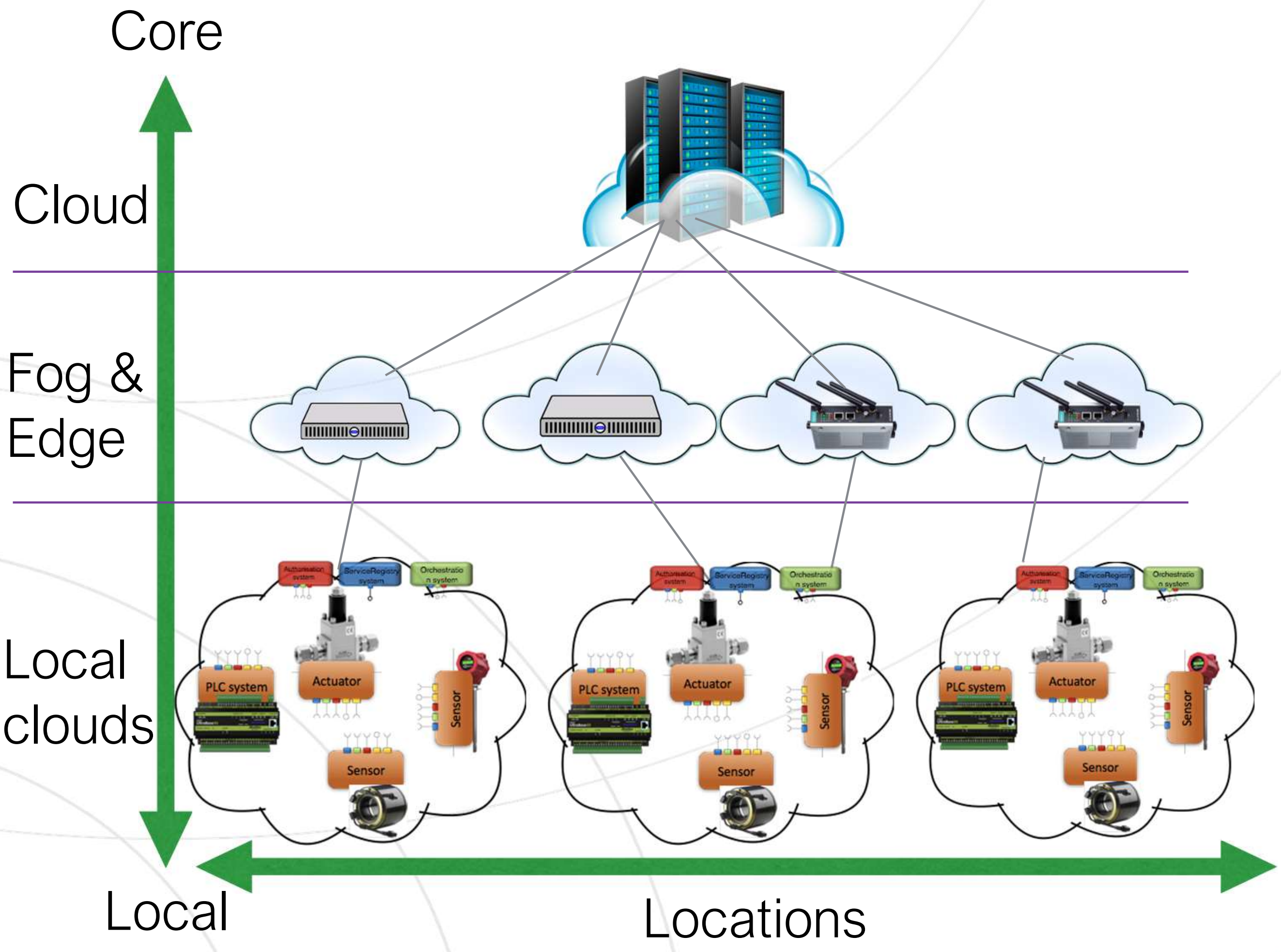

Core
Cloud
Fog &
Edge
Local
Locations

Local Cloud Meeting Automation Requirements

- **Automation is local**
- **Local clouds shall provide**
  - A protective fence
    - Against external communication
  - Inter cloud service exchange

  - Thus protecting sensitive automation operations as
    - Real time closed control loops
    - Safety critical operations





Core

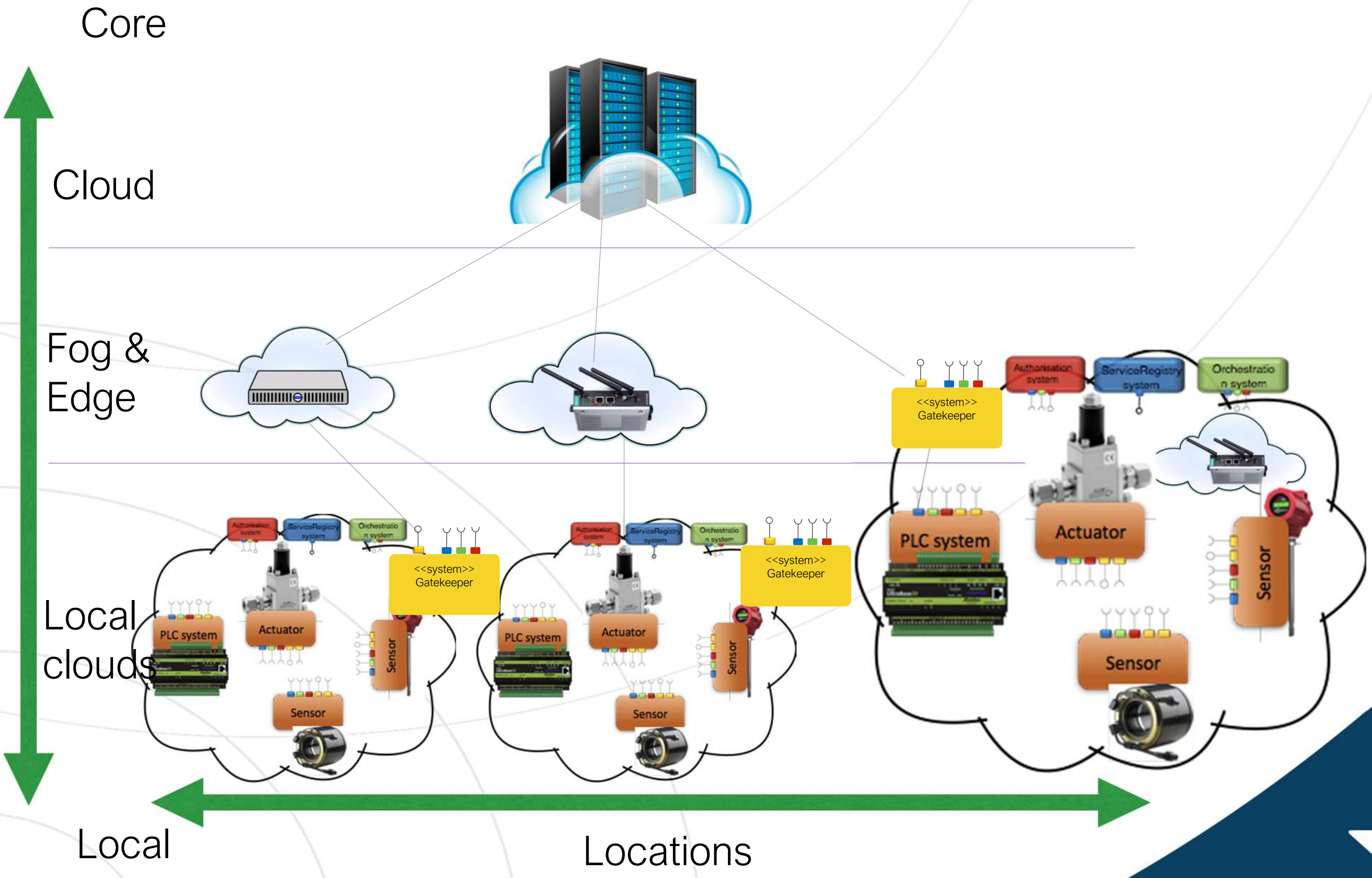
Cloud

Fog & Edge

Local clouds

Local

Locations



Core

Cloud

Fog & Edge

Local clouds

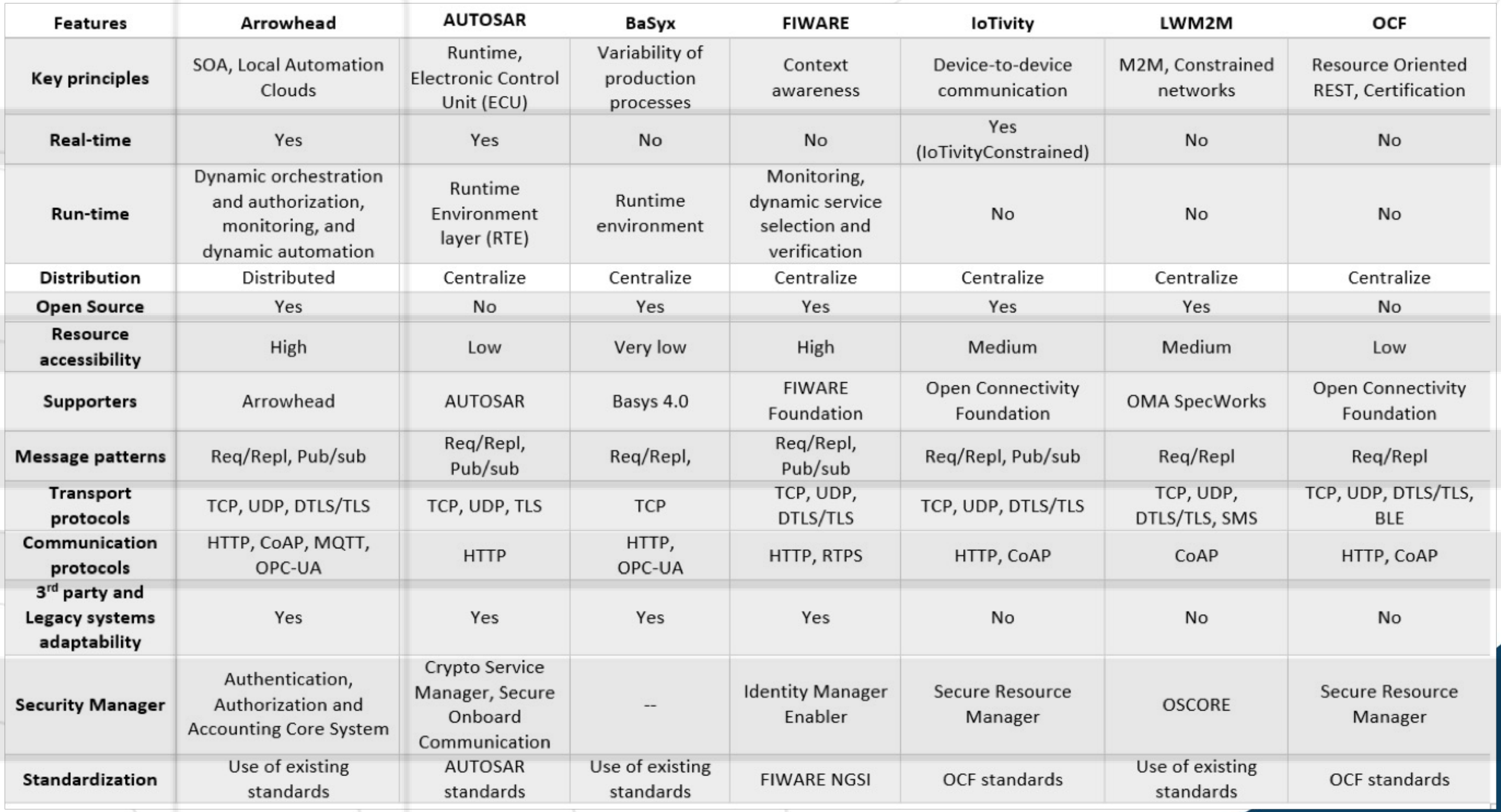
Local

Locations

www.arrowhead.eu

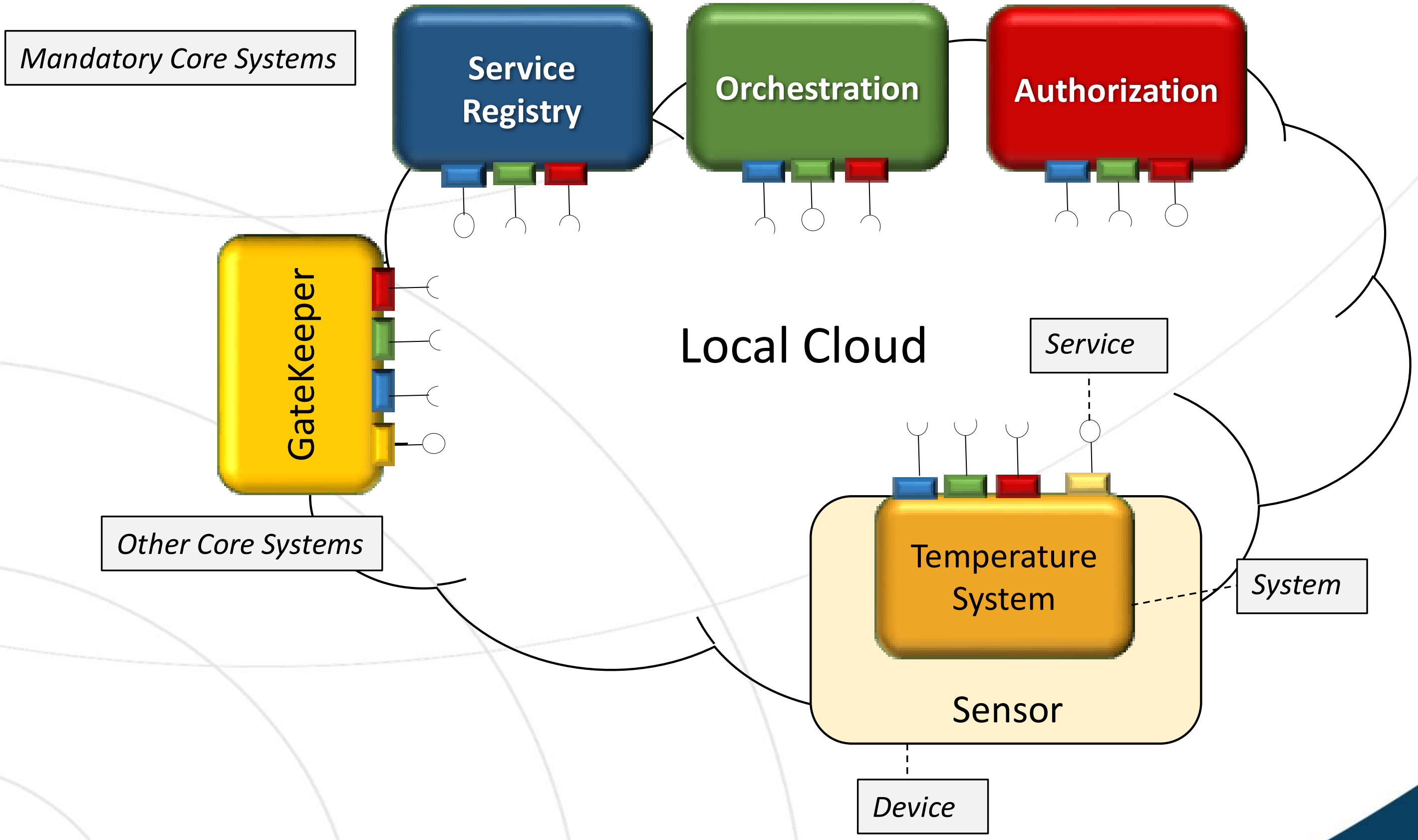
ARROWHEAD
fPVN

IoT-SoS Architectures & Platforms

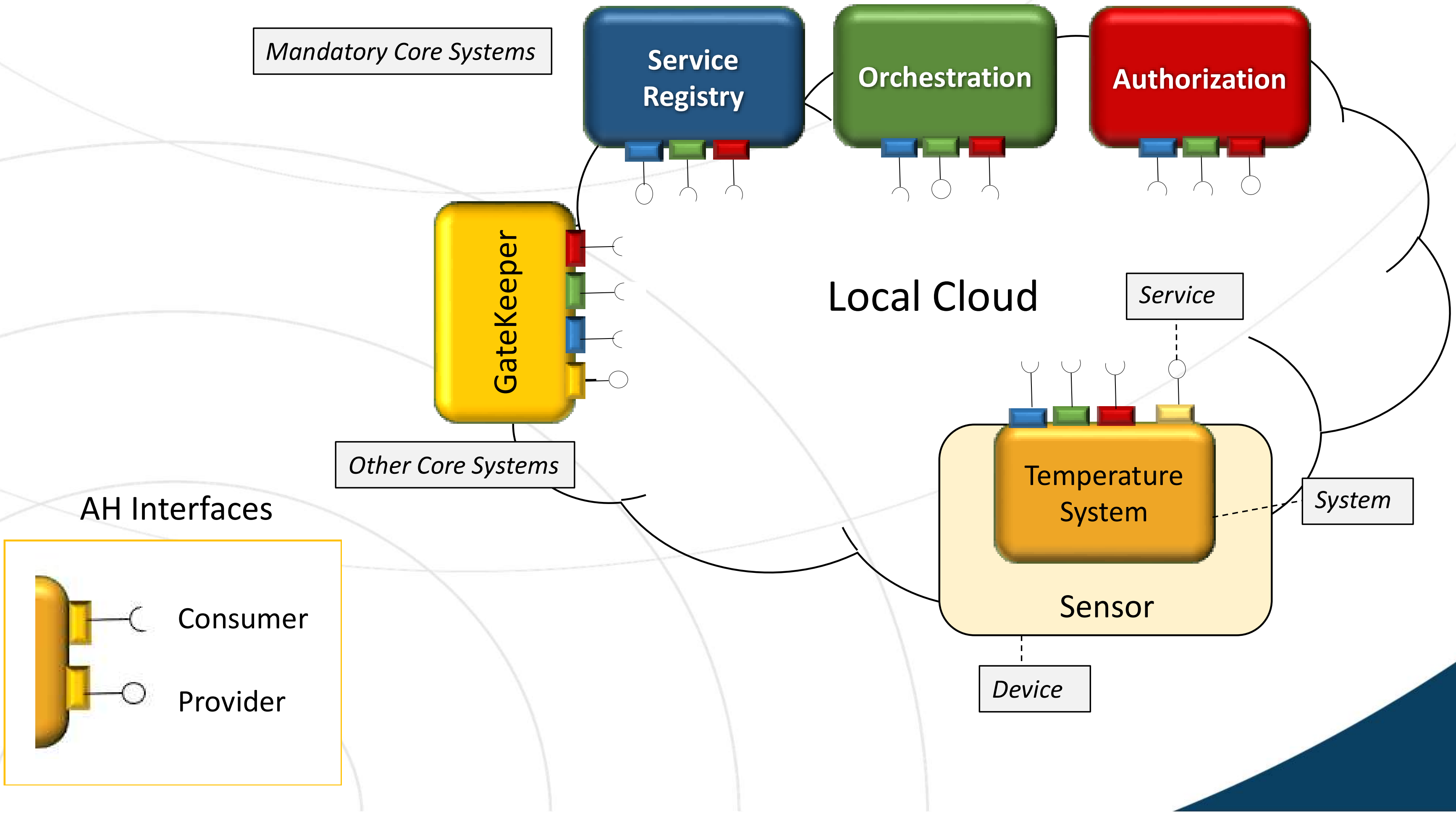


Features	Arrowhead	AUTOSAR	BaSyx	FIWARE	IoTivity	LWM2M	OCF
Key principles	SOA, Local Automation Clouds	Runtime, Electronic Control Unit (ECU)	Variability of production processes	Context awareness	Device-to-device communication	M2M, Constrained networks	Resource Oriented REST, Certification
Real-time	Yes	Yes	No	No	Yes (IoTivityConstrained)	No	No
Run-time	Dynamic orchestration and authorization, monitoring, and dynamic automation	Runtime Environment layer (RTE)	Runtime environment	Monitoring, dynamic service selection and verification	No	No	No
Distribution	Distributed	Centralize	Centralize	Centralize	Centralize	Centralize	Centralize
Open Source	Yes	No	Yes	Yes	Yes	Yes	No
Resource accessibility	High	Low	Very low	High	Medium	Medium	Low
Supporters	Arrowhead	AUTOSAR	Basys 4.0	FIWARE Foundation	Open Connectivity Foundation	OMA SpecWorks	Open Connectivity Foundation
Message patterns	Req/Repl, Pub/sub	Req/Repl, Pub/sub	Req/Repl,	Req/Repl, Pub/sub	Req/Repl, Pub/sub	Req/Repl	Req/Repl
Transport protocols	TCP, UDP, DTLS/TLS	TCP, UDP, TLS	TCP	TCP, UDP, DTLS/TLS	TCP, UDP, DTLS/TLS	TCP, UDP, DTLS/TLS, SMS	TCP, UDP, DTLS/TLS, BLE
Communication protocols	HTTP, CoAP, MQTT, OPC-UA	HTTP	HTTP, OPC-UA	HTTP, RTPS	HTTP, CoAP	CoAP	HTTP, CoAP
3rd party and Legacy systems adaptability	Yes	Yes	Yes	Yes	No	No	No
Security Manager	Authentication, Authorization and Accounting Core System	Crypto Service Manager, Secure Onboard Communication	--	Identity Manager Enabler	Secure Resource Manager	OSCORE	Secure Resource Manager
Standardization	Use of existing standards	AUTOSAR standards	Use of existing standards	FIWARE NGSI	OCF standards	Use of existing standards	OCF standards

Arrowhead Framework Elements



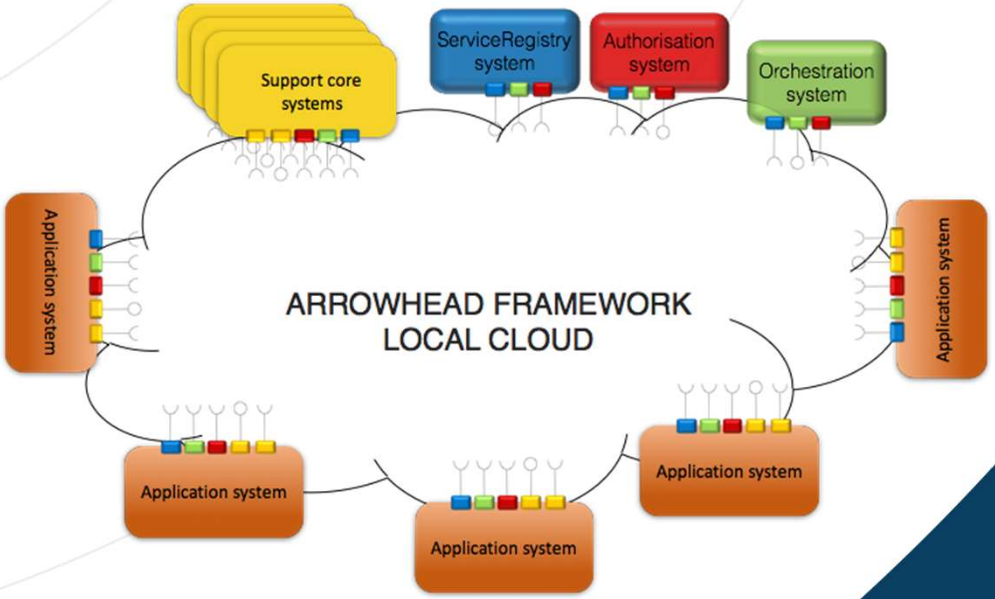
Arrowhead Framework elements



ARROWHEAD
fPVN

Local Cloud Key Properties

- Self-contained
- Provide a strong security fence to external networks
- Interoperability between systems within a local cloud is established through services of information ex-change
- System of Systems integration
- Automation support – both design- and runtime
- Security in relation to bootstrapping, software up- date, and communication in general
- Inter-cloud service exchanges



Mandatory Core Systems And Services

ServiceRegistry system

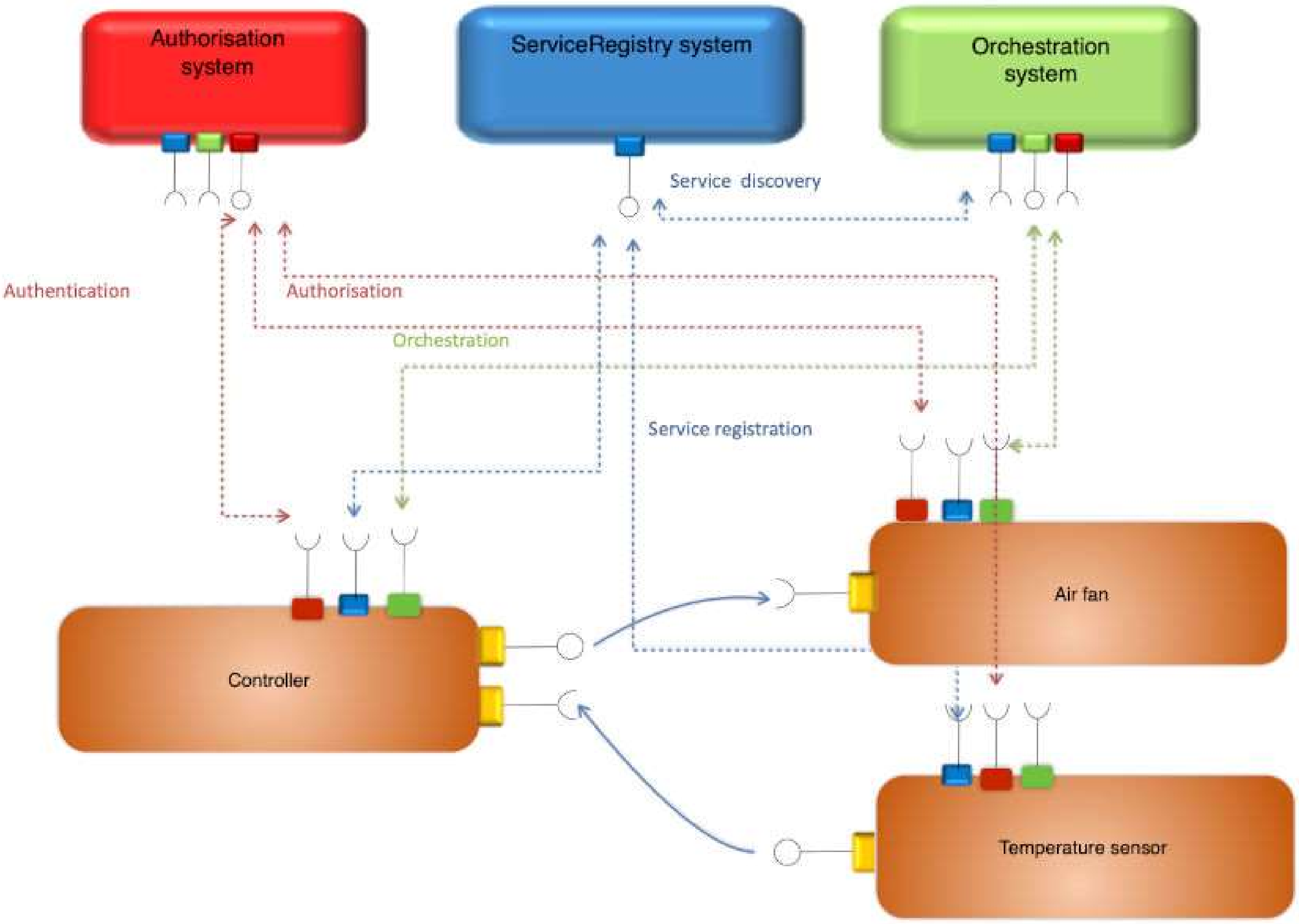
- Keeps track of all active services produced within a local cloud.

Authorization system

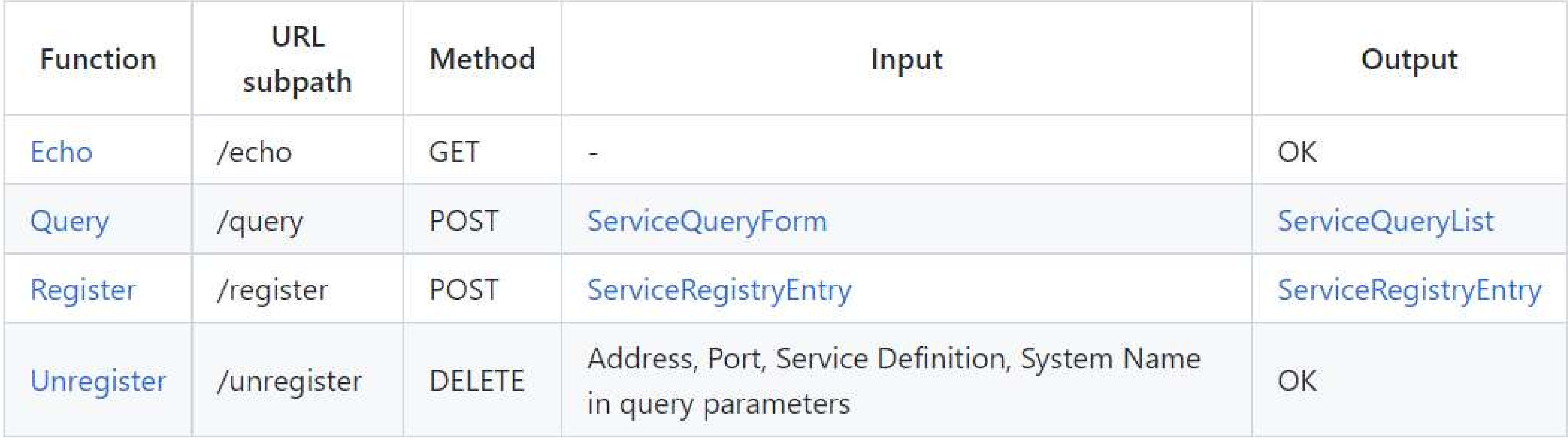
- Provides Authentication, Authorization and optionally Accounting of a system consuming a produced service.

Orchestration system

- Provides a mechanism for distributing orchestration rules and service consumption patterns.


Authorisation system
ServiceRegistry system
Orchestration system
Service discovery
Authentication
Authorisation
Orchestration
Service registration
Air fan
Controller
Temperature sensor

Understanding Service Registration



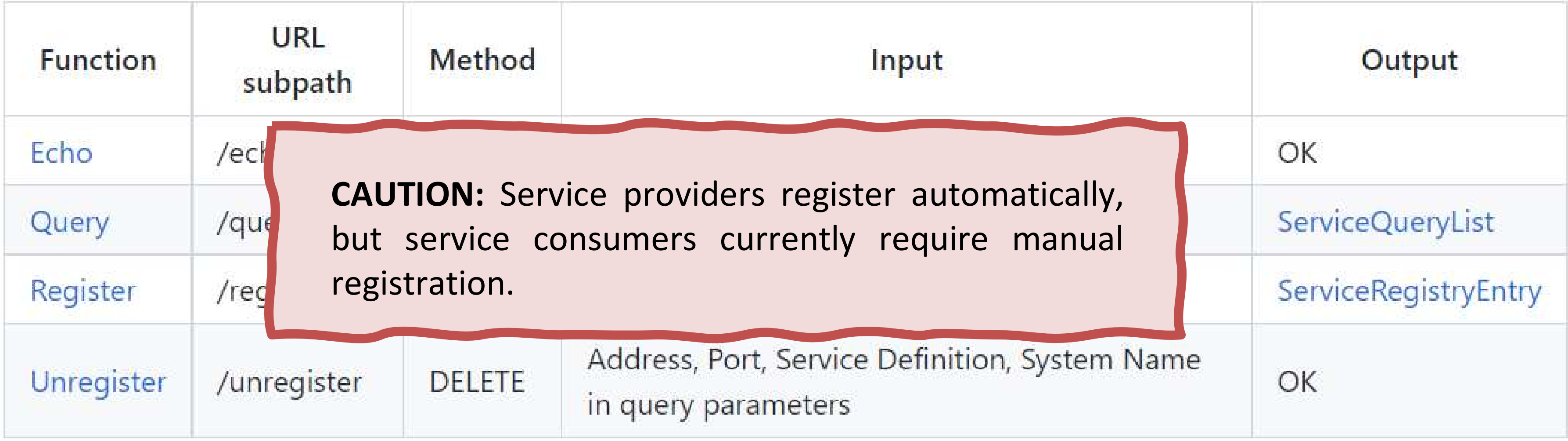
Function	URL subpath	Method	Input	Output
Echo	/echo	GET	-	OK
Query	/query	POST	ServiceQueryForm	ServiceQueryList
Register	/register	POST	ServiceRegistryEntry	ServiceRegistryEntry
Unregister	/unregister	DELETE	Address, Port, Service Definition, System Name in query parameters	OK

The **query** method is used to find and translate a symbolic service name into a physical endpoint, for example an IP address and a port.

The **register** method is used to register services. The services will contain various metadata as well as a physical endpoint.

The **unregister** method is used to unregister service instances that were previously registered in the Registry.

Understanding Service Registration



Function	URL subpath	Method	Input	Output
Echo	/ech			OK
Query	/que			ServiceQueryList
Register	/reg			ServiceRegistryEntry
Unregister	/unregister	DELETE	Address, Port, Service Definition, System Name in query parameters	OK

CAUTION: Service providers register automatically, but service consumers currently require manual registration.

The **query** method is used to find and translate a symbolic service name into a physical endpoint, for example an IP address and a port.

The **register** method is used to register services. The services will contain various metadata as well as a physical endpoint.

The **unregister** method is used to unregister service instances that were previously registered in the Registry.

Understanding the Orchestration

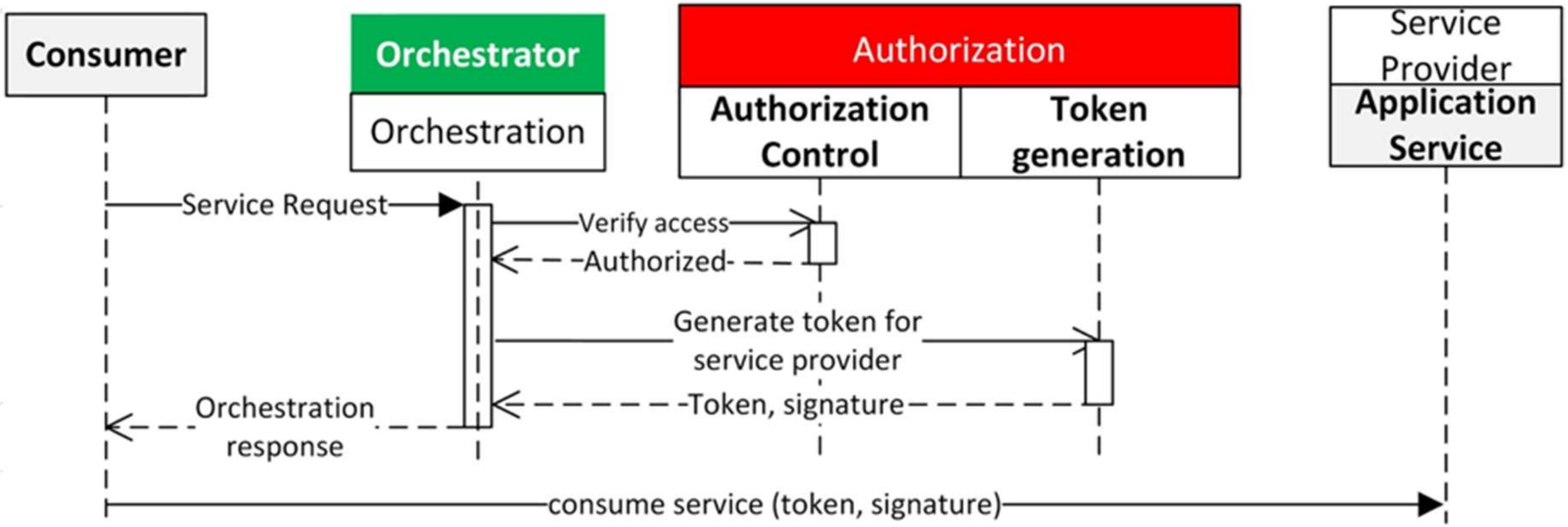
- **Store Orchestration:**
  Utilize the storage information from the database, focusing on the PROVIDER-CONSUMER-SERVICE triplets.

- **Dynamic Orchestration:**
  Query the registry based on the service request.

Orchestration Flags

- ***matchmaking***: the service automatically selects exactly one provider from the appropriate providers (if any),
- ***metadataSearch***: query in the Service Registry uses metadata filtering,
- ***onlyPreferred***: the service filters the results with the specified provider list,
- ***pingProviders***: the service checks whether the returning providers are online and remove the unaccessible ones from the results,
- ***overrideStore***: Services uses dynamic orchestration if this flag is true, otherwise it uses the orchestration store,
- ***enableInterCloud***: the service can search another clouds for providers if none of the local cloud providers match the requirements,
- ***triggerInterCloud***: the service skipped the search in the local cloud and tries to find providers in other clouds instead

Understanding Authorization



Certificates

https://github.com/eclipse-arrowhead/core-java-spring/wiki

Arrowhead Framework's security is relying on SSL Certificate Trust Chains. The Arrowhead trust chain consists of three level:

1. Master certificate: arrowhead.eu
2. Cloud certificate: my_cloud.my_company.arrowhead.eu
3. Client certificate: my_client.my_cloud.my_company.arrowhead.eu

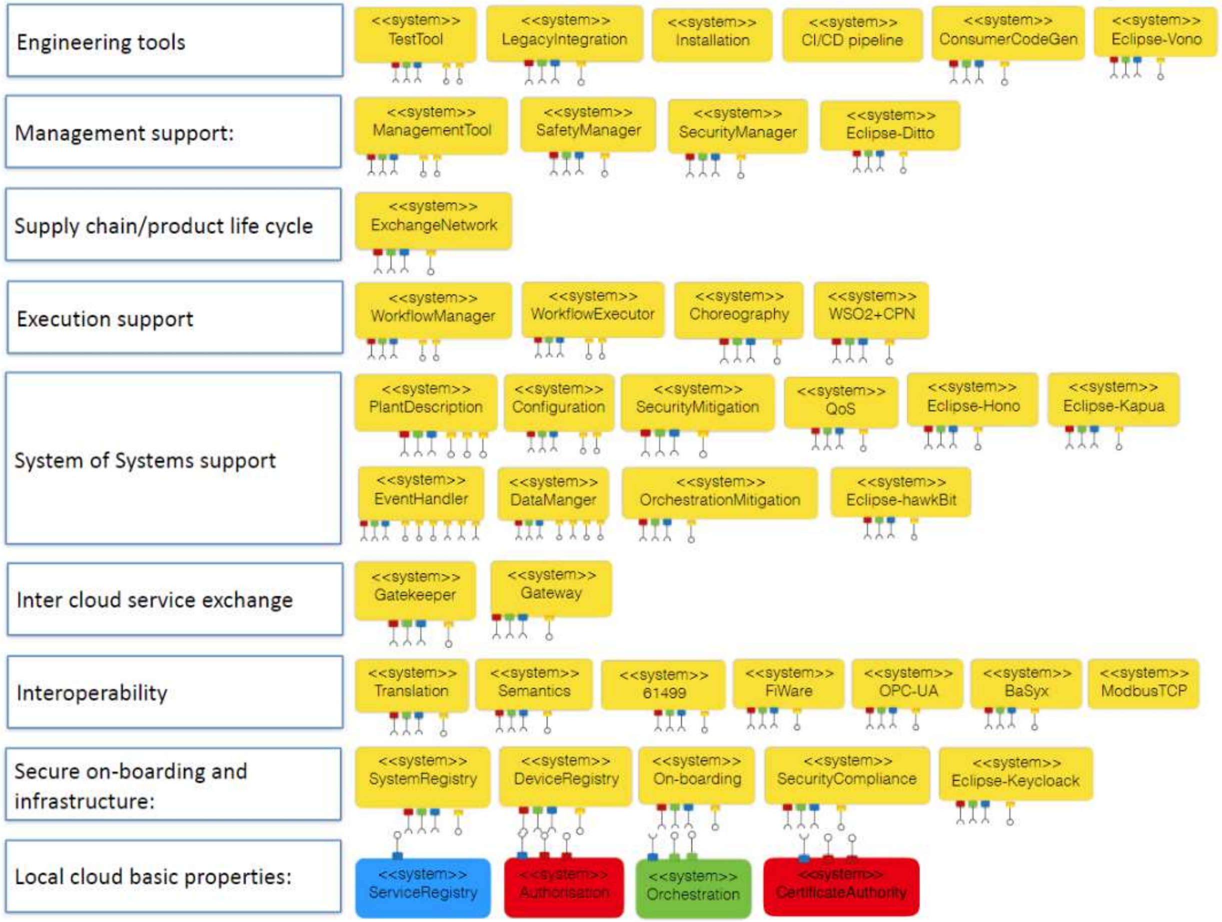
The Key-Store

The Key-Store is intended to store the certificates and/or key-pair certificates.

The Trust-Store

The Trust-Store is containing those certificates, what the web-server considers as trusted ones.

Support Systems



Engineering tools	<<system>> TestTool <<system>> LegacyIntegration <<system>> Installation <<system>> CI/CD pipeline <<system>> ConsumerCodeGen <<system>> Eclipse-Vono
Management support:	<<system>> ManagementTool <<system>> SafetyManager <<system>> SecurityManager <<system>> Eclipse-Ditto
Supply chain/product life cycle	<<system>> ExchangeNetwork
Execution support	<<system>> WorkflowManager <<system>> WorkflowExecutor <<system>> Choreography <<system>> WSO2+CPN
System of Systems support	<<system>> PlantDescription <<system>> Configuration <<system>> SecurityMitigation <<system>> QoS <<system>> Eclipse-Hono <<system>> Eclipse-Kapua <<system>> EventHandler <<system>> DataManger <<system>> OrchestrationMitigation <<system>> Eclipse-hawkBit
Inter cloud service exchange	<<system>> Gatekeeper <<system>> Gateway
Interoperability	<<system>> Translation <<system>> Semantics <<system>> 61499 <<system>> FiWare <<system>> OPC-UA <<system>> BaSyx <<system>> ModbusTCP
Secure on-boarding and infrastructure:	<<system>> SystemRegistry <<system>> DeviceRegistry <<system>> On-boarding <<system>> SecurityCompliance <<system>> Eclipse-Keycloack
Local cloud basic properties:	<<system>> ServiceRegistry <<system>> Authorisation <<system>> Orchestration <<system>> CertificateAuthority

ARROWHEAD
fPVN

Resources

GitHub: https://github.com/eclipse-arrowhead/core-java-spring
https://github.com/arrowhead-f

Wiki: https://github.com/eclipse-arrowhead/core-java-spring/wiki

Papers

Book → IoT Automation: Arrowhead Framework

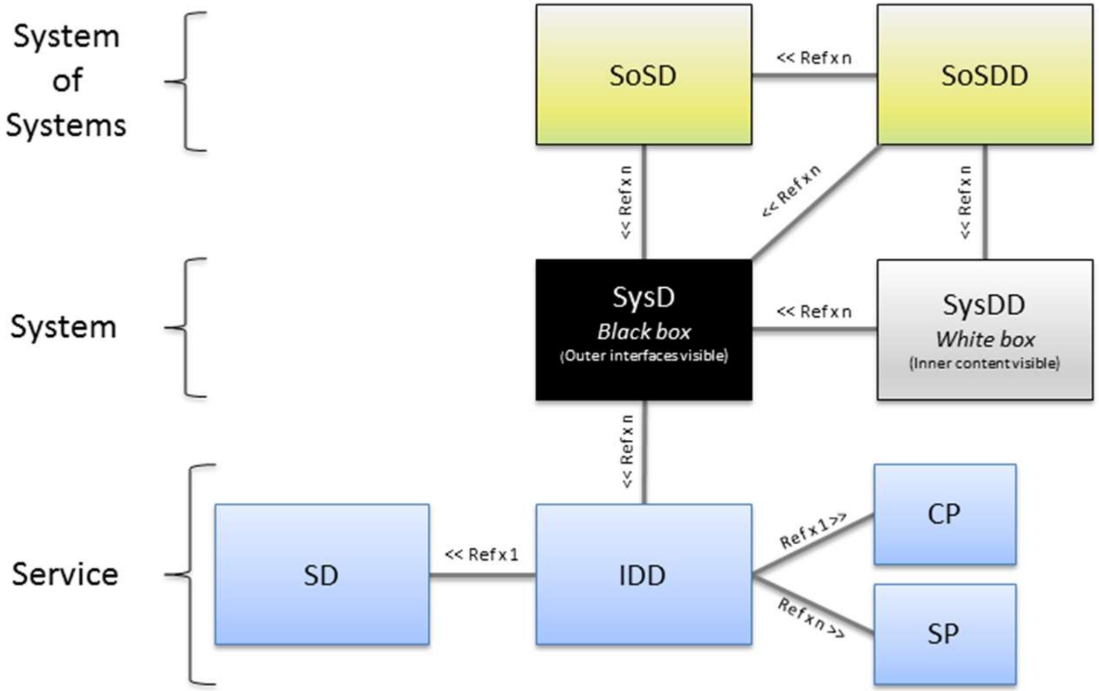
It can be installed using:

- Docker
- Debian Installers
- Source code (MySQL and Maven) → Windows, Mac OS and Linux based OS

Documentation Structure

A three-level documentation structure is defined:

- System of Systems level
- System level
- Service level.



Running the Mandatory Core Systems

1. Download /clone the git folders.
2. Core systems folder → built the projects: “mvn clean install”
3. Run core systems

   Minimal set:

   1st Service Registry

   2nd Authorization

   3rd Orchestration

   “java –jar ---------.jar”

Security Insights by Prof. Markus Tauber

Hands-on Tutorial

Running the Mandatory Core Systems

1. Download /clone the git folders.
2. Core systems folder → built the projects: “mvn clean install”
3. Run core systems

   Minimal set:

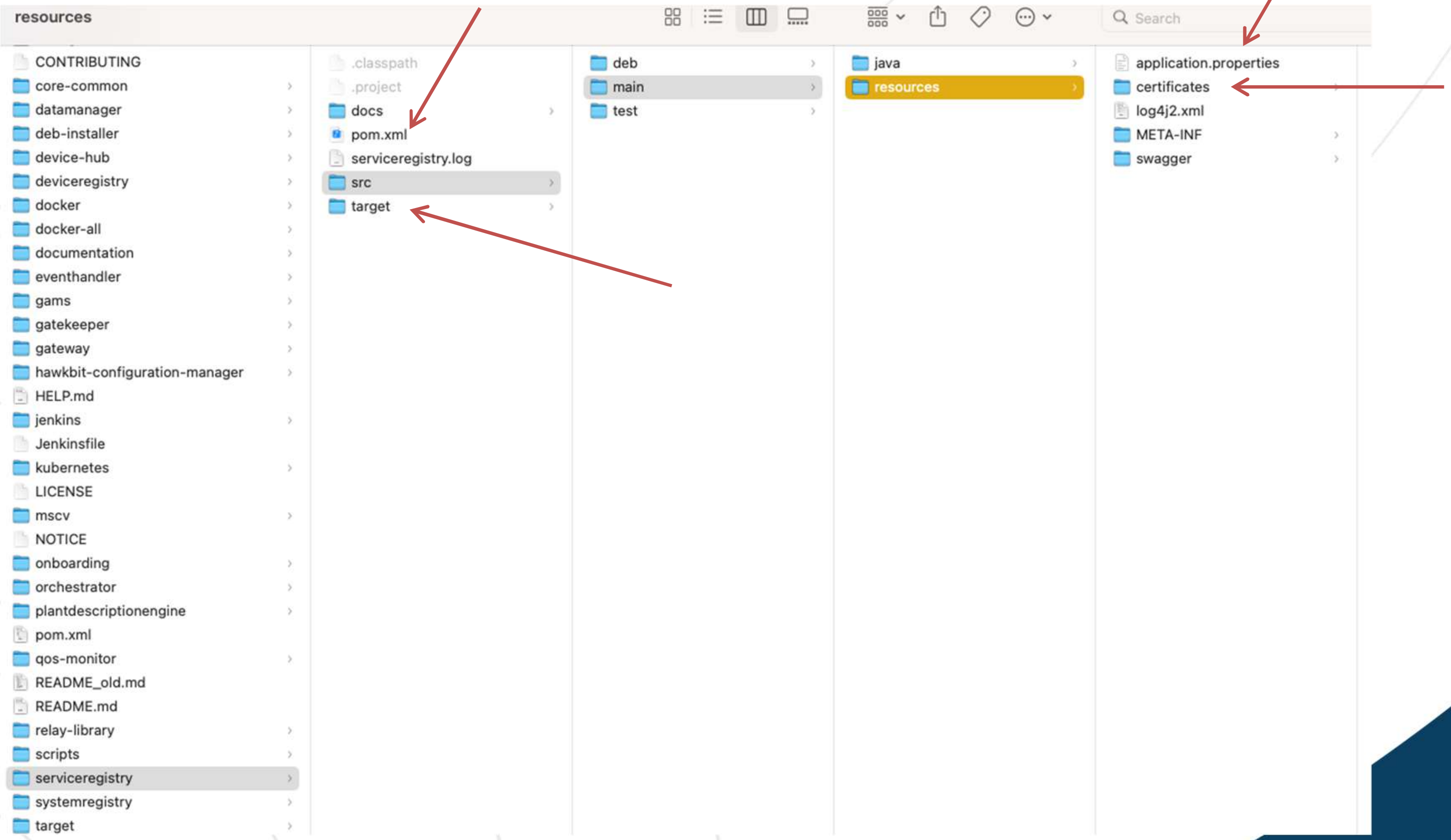
   1st Service Registry

   2nd Authorization

   3rd Orchestration

   “java –jar ---------.jar”

Getting Familiar with the Project



resources

Search

CONTRIBUTING
core-common
datamanager
deb-installer
device-hub
deviceregistry
docker
docker-all
documentation
eventhandler
gams
gatekeeper
gateway
hawkbit-configuration-manager
HELP.md
jenkins
Jenkinsfile
kubernetes
LICENSE
mscv
NOTICE
onboarding
orchestrator
plantdescriptionengine
pom.xml
qos-monitor
README_old.md
README.md
relay-library
scripts
serviceregistry
systemregistry
target

.classpath
.project
docs
pom.xml
serviceregistry.log
src
target

deb
main
test

java
resources

application.properties
certificates
log4j2.xml
META-INF
swagger

Configuring the Database!

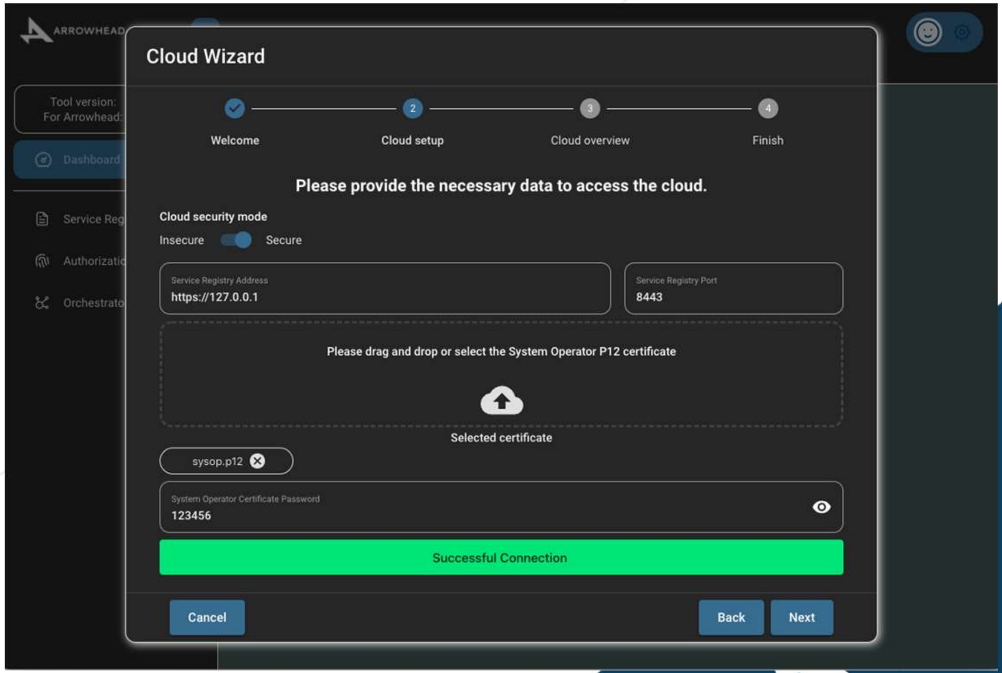
1st Scripts folder→ run create_arrowhead_tables.sql
2nd Creare users and set priviledges (follow create_empty_arrowhead_db)

Arrowhead Management Tool Chain

https://www.aitia.ai/products/arrowhead-tools/

- Arrowhead Certificate Generator
- Arrowhead Framework Installer
- Arrowhead Management Tool

VIDEO TUTORIAL
https://www.youtube.com/watch?v=e5zrY1aqgBQ&t=1916s



Cloud Wizard

Please provide the necessary data to access the cloud.

Cloud security mode: Insecure / Secure

Service Registry Address: https://127.0.0.1

Service Registry Port: 8443

Please drag and drop or select the System Operator P12 certificate

Selected certificate: sysop.p12

System Operator Certificate Password: 123456

Successful Connection

ARROWHEAD
fPVN

Running An Example - Steps

1. Demo folder → built the projects: “mvn install”
2. Demo folder → run the jar generated (Script or individual jars)

   → ERROR → The database has to be configured!

1. Open the MySQL Arrowhead database and configure the database → next slide
2. Run again!→ Working!

Pre-requisites:

- Java 11
- Maven
- MySQL → Arrowhead database script running

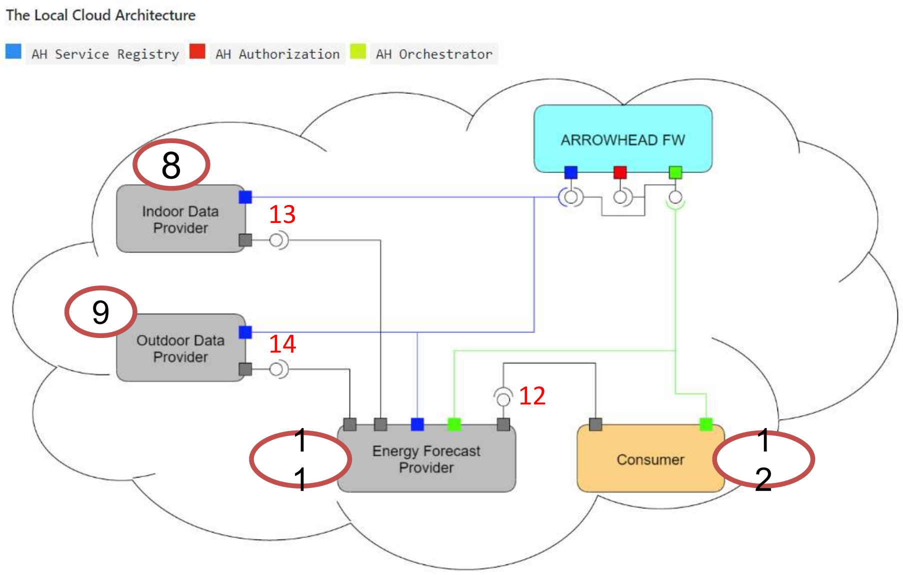
Running An Example - Steps

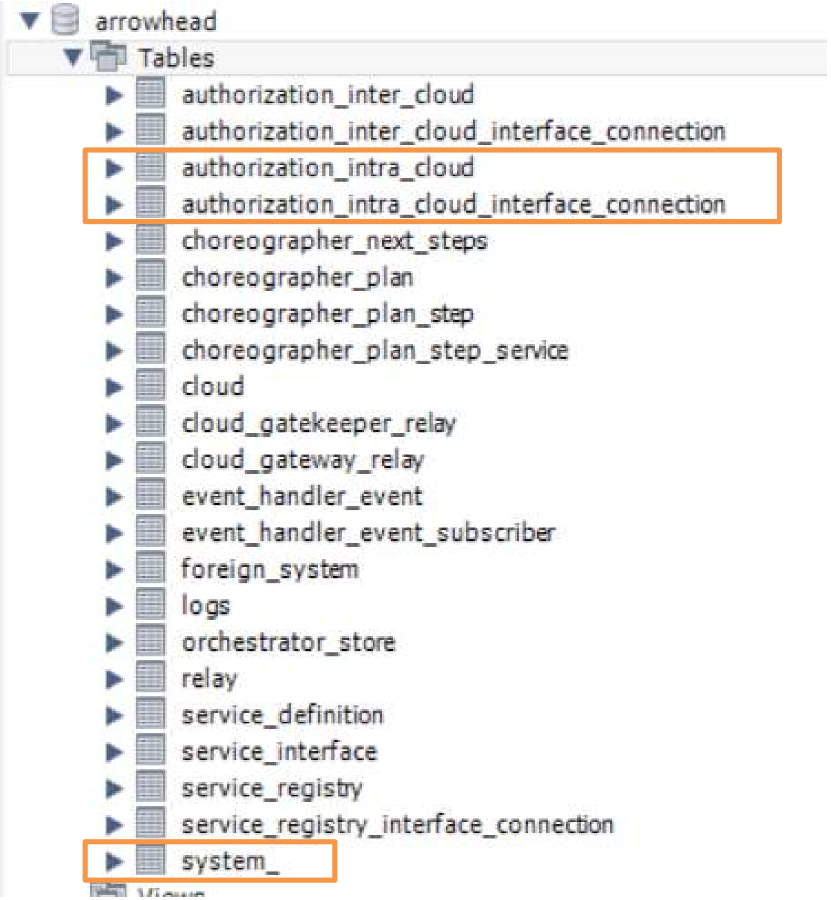
DATABASE CONFIGURATION

Register the consumer systems manually.

Register the authorization rules in the:

- Intra cloud authorization.
- Intra cloud interface authorization.




▼ arrowhead
▼ Tables
▶ authorization_inter_cloud
▶ authorization_inter_cloud_interface_connection
▶ authorization_intra_cloud
▶ authorization_intra_cloud_interface_connection
▶ choreographer_next_steps
▶ choreographer_plan
▶ choreographer_plan_step
▶ choreographer_plan_step_service
▶ cloud
▶ cloud_gatekeeper_relay
▶ cloud_gateway_relay
▶ event_handler_event
▶ event_handler_event_subscriber
▶ foreign_system
▶ logs
▶ orchestrator_store
▶ relay
▶ service_definition
▶ service_interface
▶ service_registry
▶ service_registry_interface_connection
▶ system_

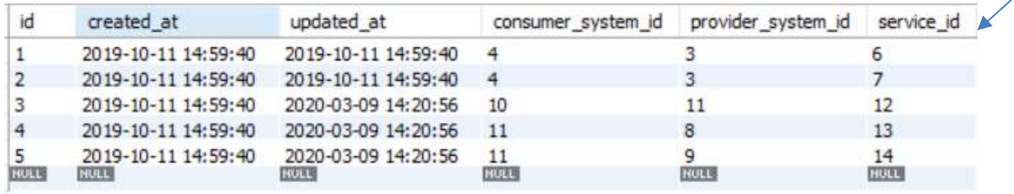
Running An Example - Steps

DATABASE CONFIGURATION

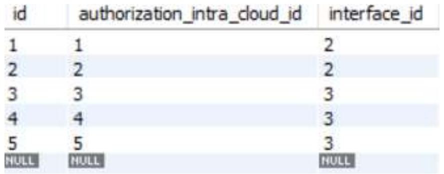
Register the consumer systems manually.

Register the authorization rules in the:

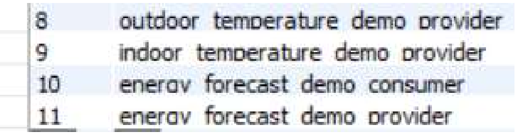
- Intra cloud authorization.
- Intra cloud interface authorization.



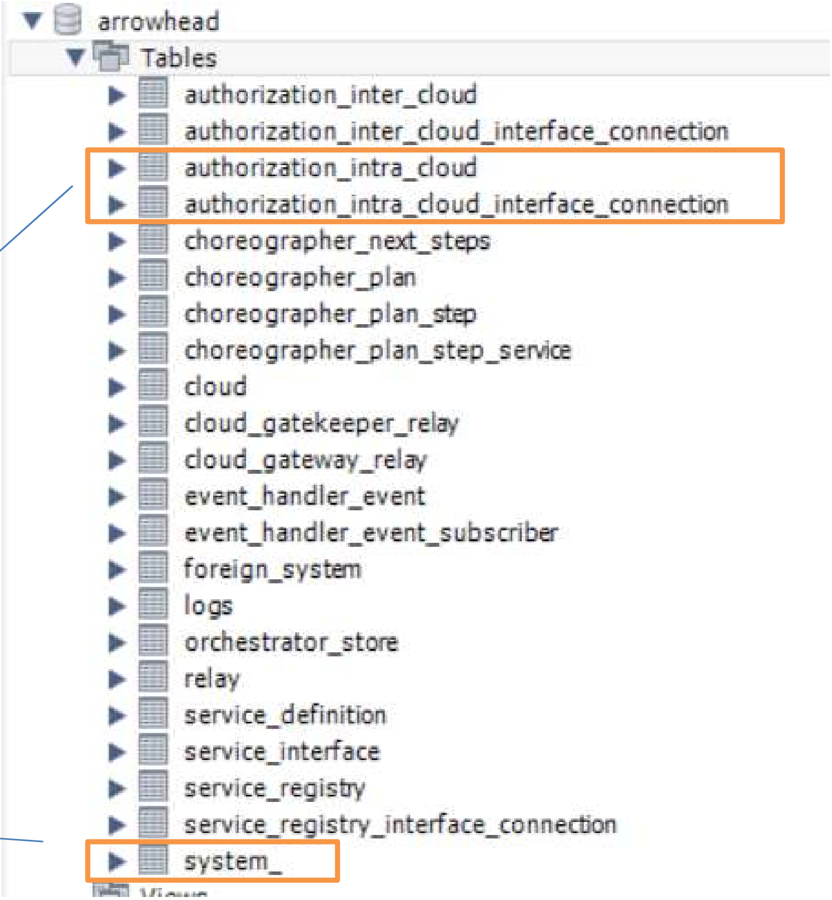
id	created_at	updated_at	consumer_system_id	provider_system_id	service_id
1	2019-10-11 14:59:40	2019-10-11 14:59:40	4	3	6
2	2019-10-11 14:59:40	2019-10-11 14:59:40	4	3	7
3	2019-10-11 14:59:40	2020-03-09 14:20:56	10	11	12
4	2019-10-11 14:59:40	2020-03-09 14:20:56	11	8	13
5	2019-10-11 14:59:40	2020-03-09 14:20:56	11	9	14
NULL	NULL	NULL	NULL	NULL	NULL



id	authorization_intra_cloud_id	interface_id
1	1	2
2	2	2
3	3	3
4	4	3
5	5	3
NULL	NULL	NULL



8	outdoor_temperature_demo_provider
9	indoor_temperature_demo_provider
10	energy_forecast_demo_consumer
11	energy_forecast_demo_provider



- arrowhead
  - Tables
    - authorization_inter_cloud
    - authorization_inter_cloud_interface_connection
    - authorization_intra_cloud
    - authorization_intra_cloud_interface_connection
    - choreographer_next_steps
    - choreographer_plan
    - choreographer_plan_step
    - choreographer_plan_step_service
    - cloud
    - cloud_gatekeeper_relay
    - cloud_gateway_relay
    - event_handler_event
    - event_handler_event_subscriber
    - foreign_system
    - logs
    - orchestrator_store
    - relay
    - service_definition
    - service_interface
    - service_registry
    - service_registry_interface_connection
    - system_

Deploying Your Own System

www.arrowhead.eu

ARROWHEAD

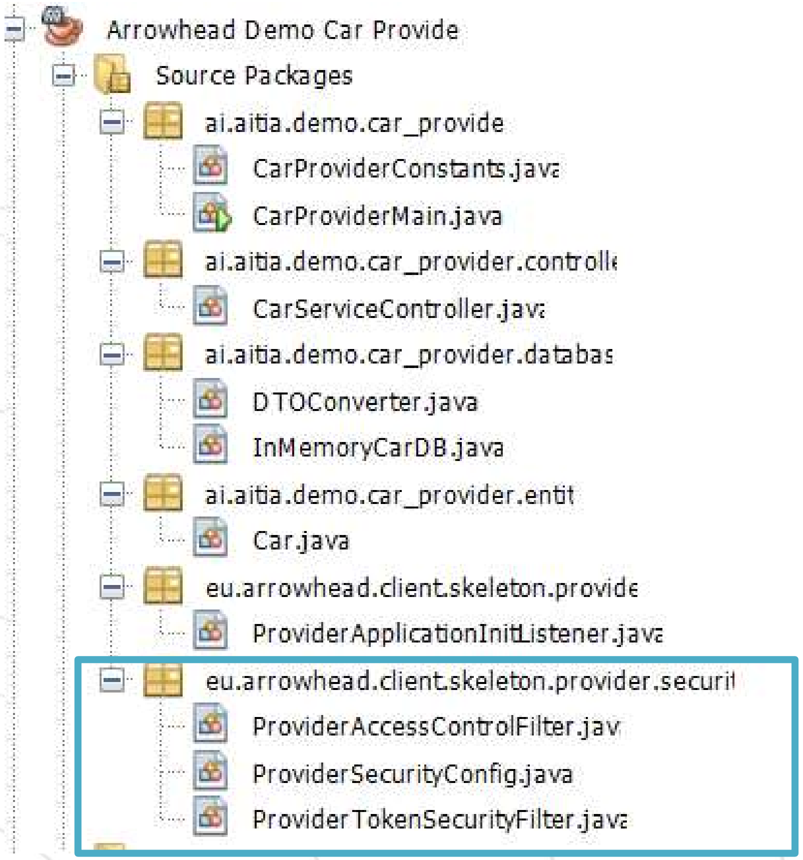
fPVN

Deploying Your Own Provider - Steps

Modify the:

1. Provider properties and constants.
2. Service Resources (Service definition, method, interfaces and metadata).
3. Customize data structures/classes and databases.
4. Certificates.

Deploying Your Own Provider - Steps

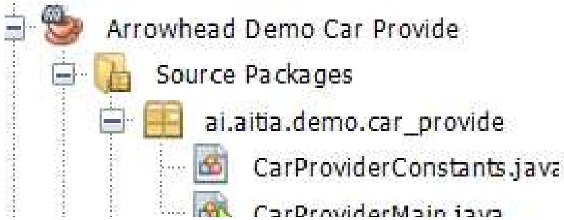


```
Arrowhead Demo Car Provide
  Source Packages
    ai.aitia.demo.car_provide
      CarProviderConstants.java
      CarProviderMain.java
    ai.aitia.demo.car_provider.controlle
      CarServiceController.java
    ai.aitia.demo.car_provider.databas
      DTOConverter.java
      InMemoryCarDB.java
    ai.aitia.demo.car_provider.entit
      Car.java
    eu.arrowhead.client.skeleton.provide
      ProviderApplicationInitListener.java
    eu.arrowhead.client.skeleton.provider.securit
      ProviderAccessControlFilter.jav
      ProviderSecurityConfig.java
      ProviderTokenSecurityFilter.java
```

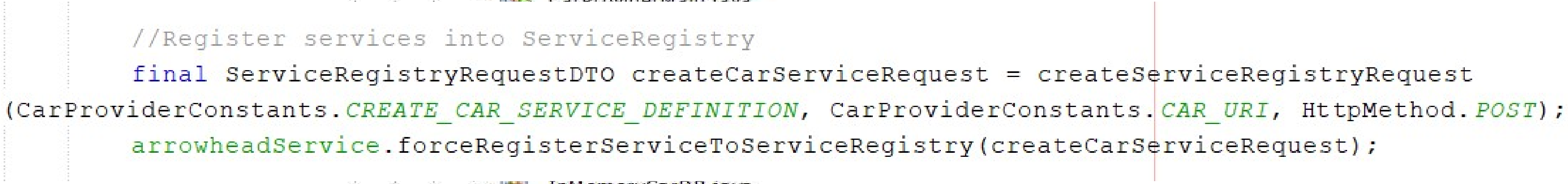
DON'T MODIFY

→ Core systems communication and security configuration

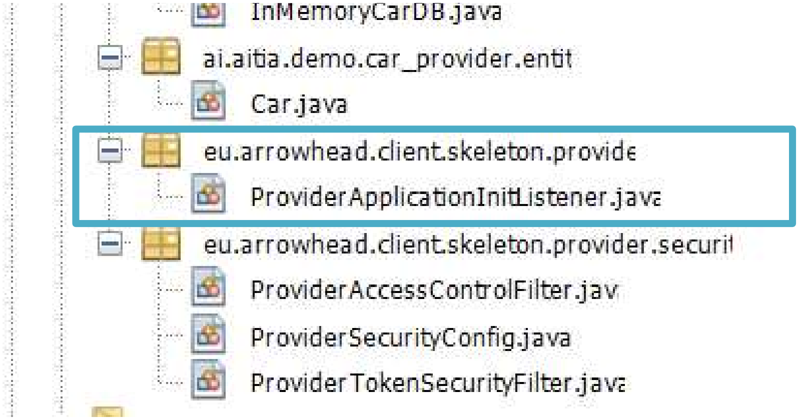
Deploying Your Own Provider - Steps



Arrowhead Demo Car Provide
Source Packages
ai.aitia.demo.car_provide
CarProviderConstants.java
CarProviderMain.java



```
        //Register services into ServiceRegistry
        final ServiceRegistryRequestDTO createCarServiceRequest = createServiceRegistryRequest
(CarProviderConstants.CREATE_CAR_SERVICE_DEFINITION, CarProviderConstants.CAR_URI, HttpMethod.POST);
        arrowheadService.forceRegisterServiceToServiceRegistry(createCarServiceRequest);
```



InMemoryCarDB.java
ai.aitia.demo.car_provider.entit
Car.java
eu.arrowhead.client.skeleton.provide
ProviderApplicationInitListener.java
eu.arrowhead.client.skeleton.provider.securit
ProviderAccessControlFilter.jav
ProviderSecurityConfig.java
ProviderTokenSecurityFilter.java

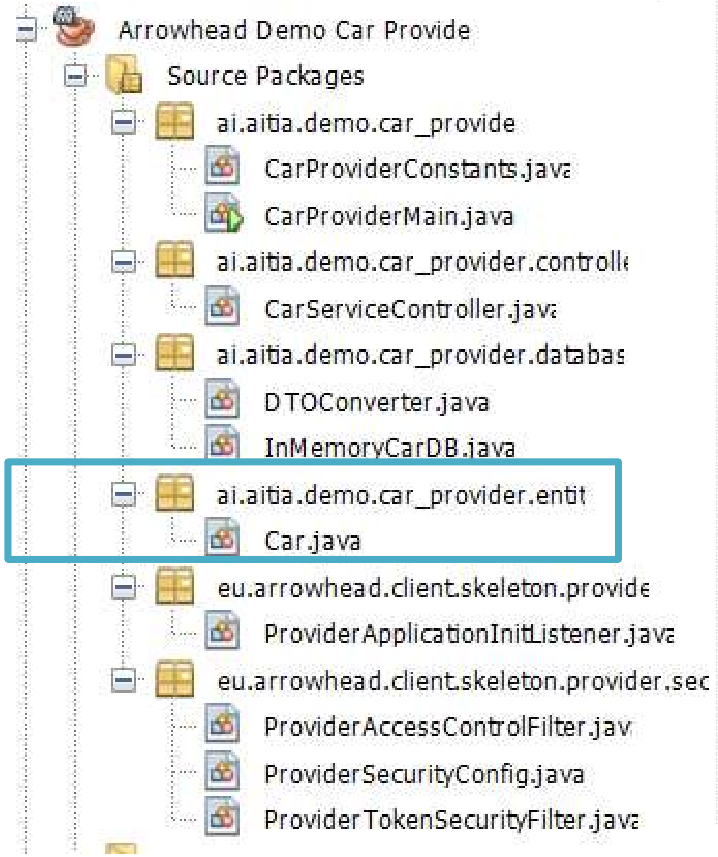
ADD SERVICE REGISTRATION

Core systems communication

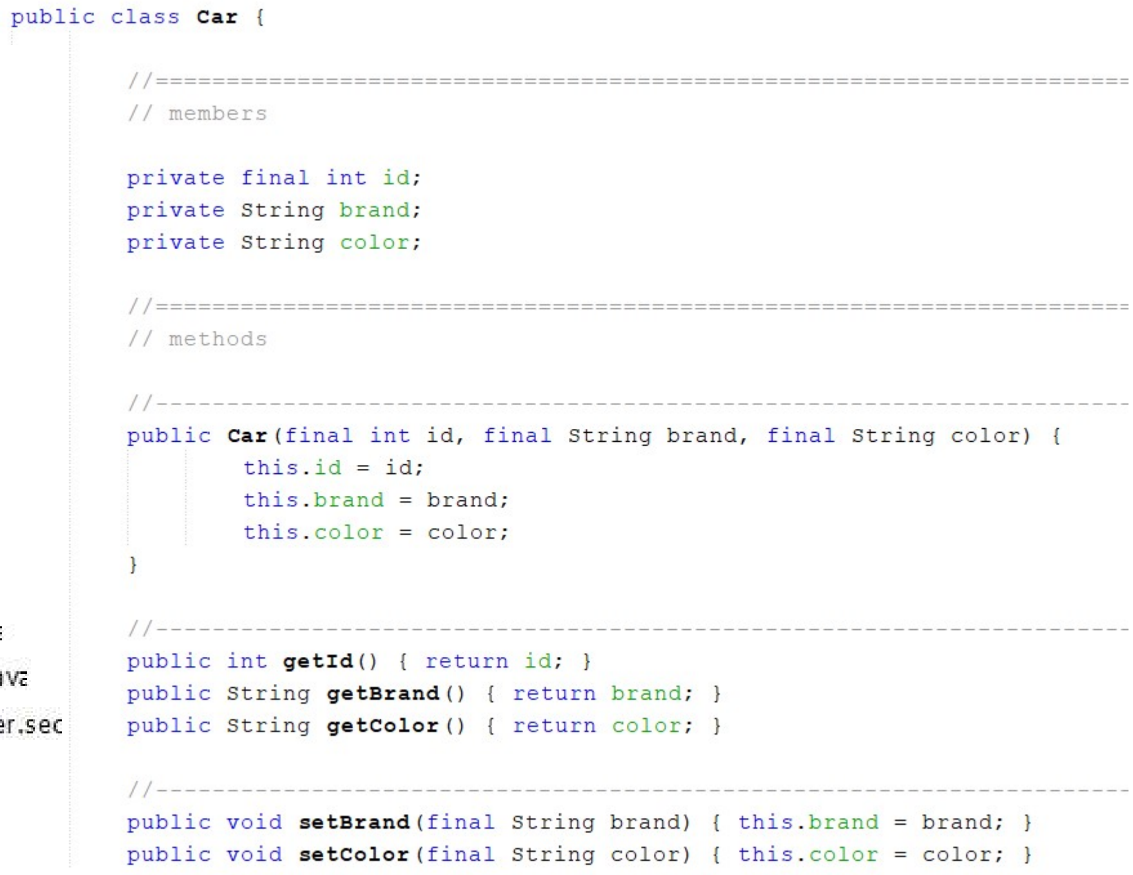
Deploying Your Own Provider - Steps

CUSTOMIZE

Object class (POJO)



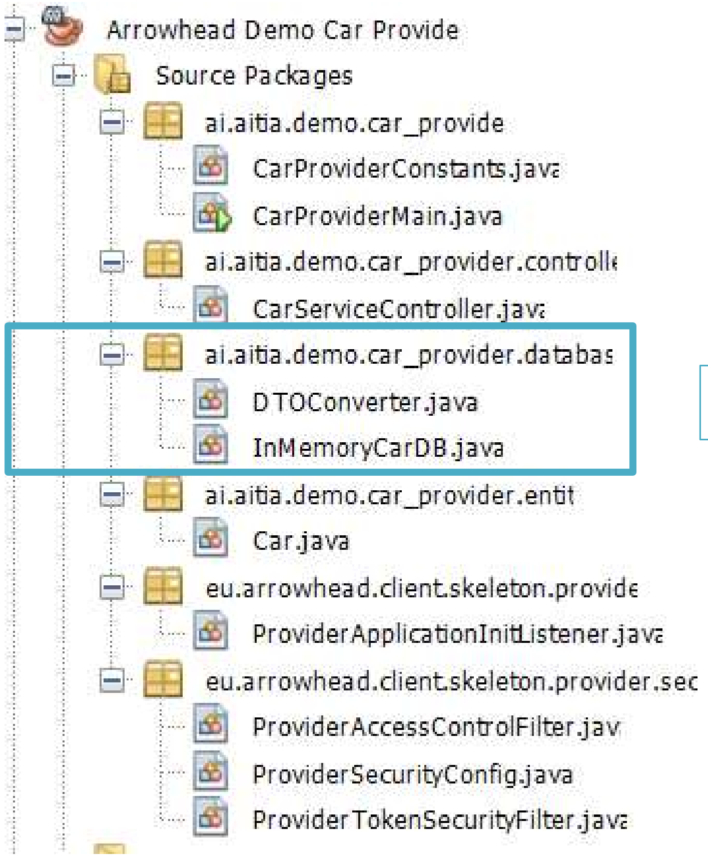
```
Arrowhead Demo Car Provide
  Source Packages
    ai.aitia.demo.car_provide
      CarProviderConstants.jav
      CarProviderMain.java
    ai.aitia.demo.car_provider.controlle
      CarServiceController.jav
    ai.aitia.demo.car_provider.databas
      DTOConverter.java
      InMemoryCarDB.java
    ai.aitia.demo.car_provider.entit
      Car.java
    eu.arrowhead.client.skeleton.provide
      ProviderApplicationInitListener.jav
    eu.arrowhead.client.skeleton.provider.sec
      ProviderAccessControlFilter.jav
      ProviderSecurityConfig.java
      ProviderTokenSecurityFilter.jav
```



```
public class Car {

    //=================================================================
    // members

    private final int id;
    private String brand;
    private String color;

    //=================================================================
    // methods

    //-----------------------------------------------------------------
    public Car(final int id, final String brand, final String color) {
        this.id = id;
        this.brand = brand;
        this.color = color;
    }

    //-----------------------------------------------------------------
    public int getId() { return id; }
    public String getBrand() { return brand; }
    public String getColor() { return color; }

    //-----------------------------------------------------------------
    public void setBrand(final String brand) { this.brand = brand; }
    public void setColor(final String color) { this.color = color; }
```

Deploying Your Own Provider - Steps

Database functions



```
Arrowhead Demo Car Provide
  Source Packages
    ai.aitia.demo.car_provide
      CarProviderConstants.java
      CarProviderMain.java
    ai.aitia.demo.car_provider.controll
      CarServiceController.java
    ai.aitia.demo.car_provider.databas
      DTOConverter.java
      InMemoryCarDB.java
    ai.aitia.demo.car_provider.entit
      Car.java
    eu.arrowhead.client.skeleton.provide
      ProviderApplicationInitListener.java
    eu.arrowhead.client.skeleton.provider.sec
      ProviderAccessControlFilter.jav
      ProviderSecurityConfig.java
      ProviderTokenSecurityFilter.java
```

CUSTOMIZE

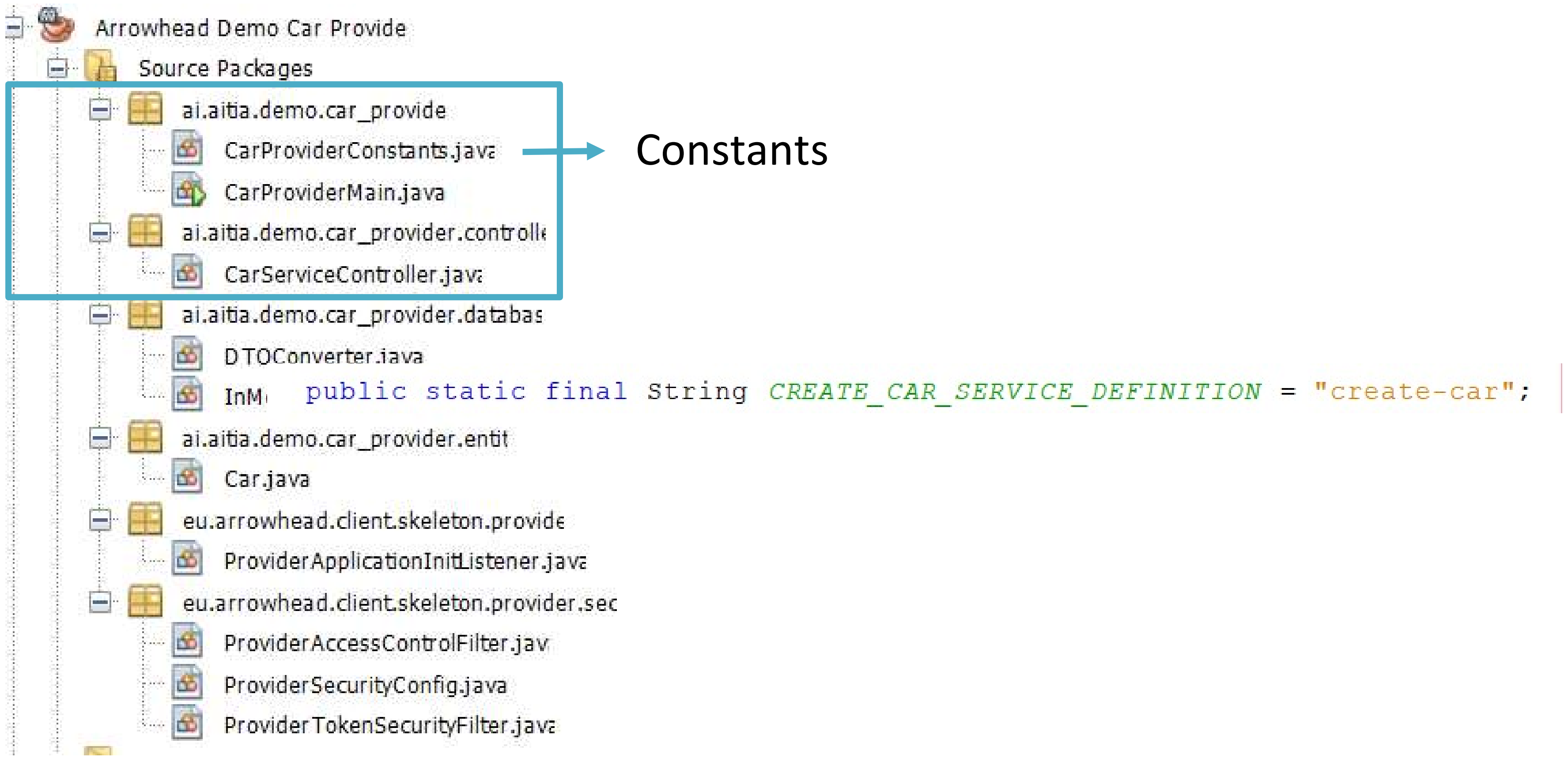
ARROWHEAD TOOLS

www.arrowhead.eu

Deploying Your Own Provider - Steps

MODIFY

Important functions



```
Arrowhead Demo Car Provide
  Source Packages
    ai.aitia.demo.car_provide
      CarProviderConstants.java
      CarProviderMain.java
    ai.aitia.demo.car_provider.controll
      CarServiceController.java
    ai.aitia.demo.car_provider.databas
      DTOConverter.java
      InM
    ai.aitia.demo.car_provider.entit
      Car.java
    eu.arrowhead.client.skeleton.provide
      ProviderApplicationInitListener.java
    eu.arrowhead.client.skeleton.provider.sec
      ProviderAccessControlFilter.jav
      ProviderSecurityConfig.java
      ProviderTokenSecurityFilter.java
```

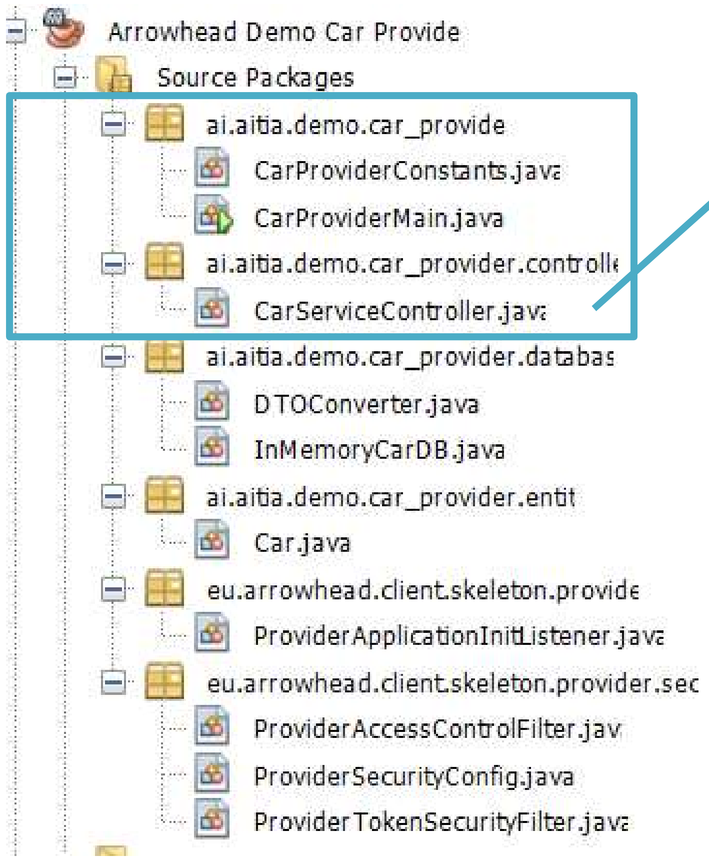
```java
public static final String CREATE_CAR_SERVICE_DEFINITION = "create-car";
```

Constants

Deploying Your Own Provider - Steps

MODIFY

Important functions



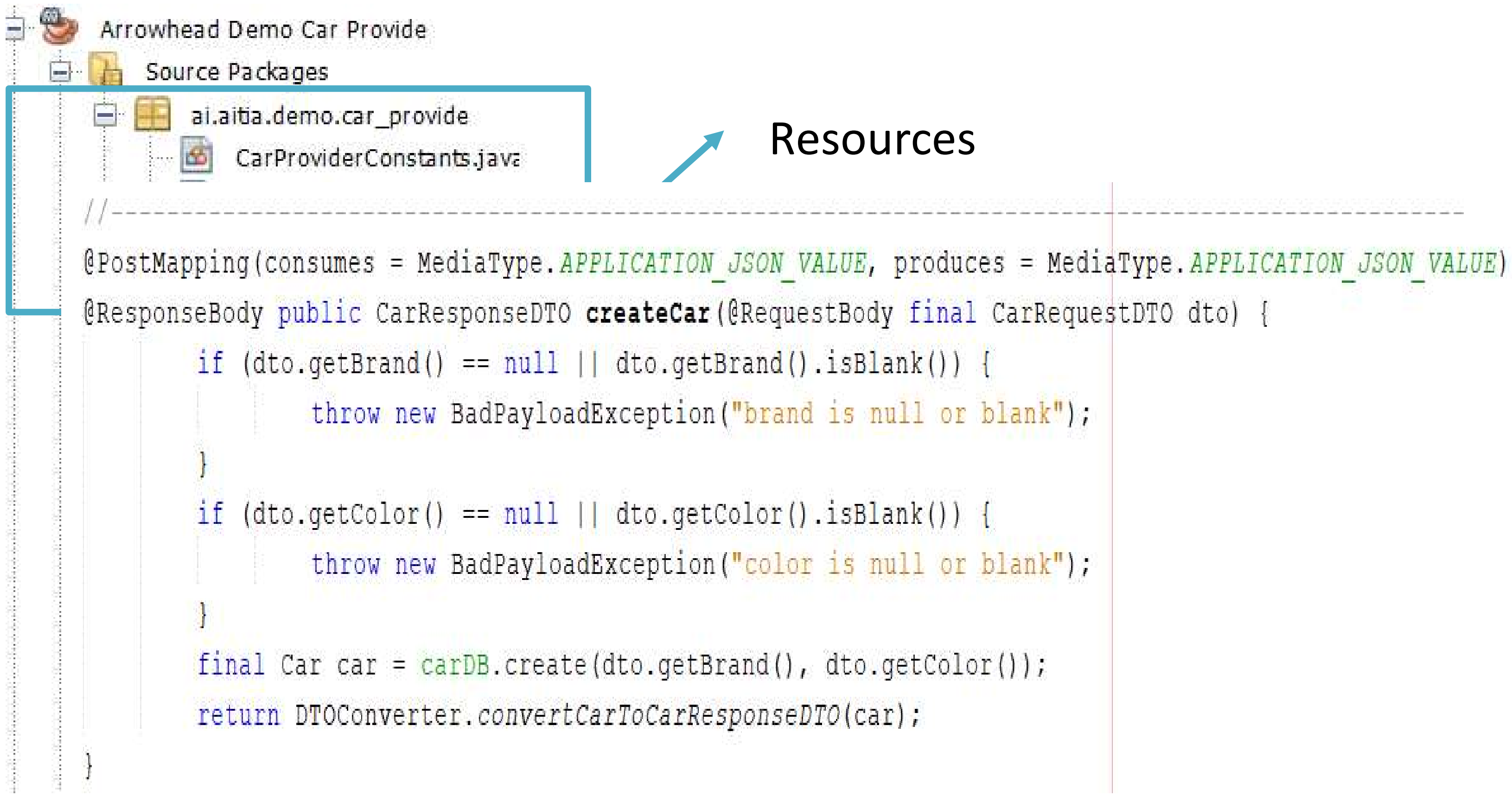
```
Arrowhead Demo Car Provide
  Source Packages
    ai.aitia.demo.car_provide
      CarProviderConstants.java
      CarProviderMain.java
    ai.aitia.demo.car_provider.controlle
      CarServiceController.java
    ai.aitia.demo.car_provider.databas
      DTOConverter.java
      InMemoryCarDB.java
    ai.aitia.demo.car_provider.entit
      Car.java
    eu.arrowhead.client.skeleton.provide
      ProviderApplicationInitListener.java
    eu.arrowhead.client.skeleton.provider.sec
      ProviderAccessControlFilter.jav
      ProviderSecurityConfig.java
      ProviderTokenSecurityFilter.java
```

Resources

Deploying Your Own Provider - Steps

MODIFY

Important functions



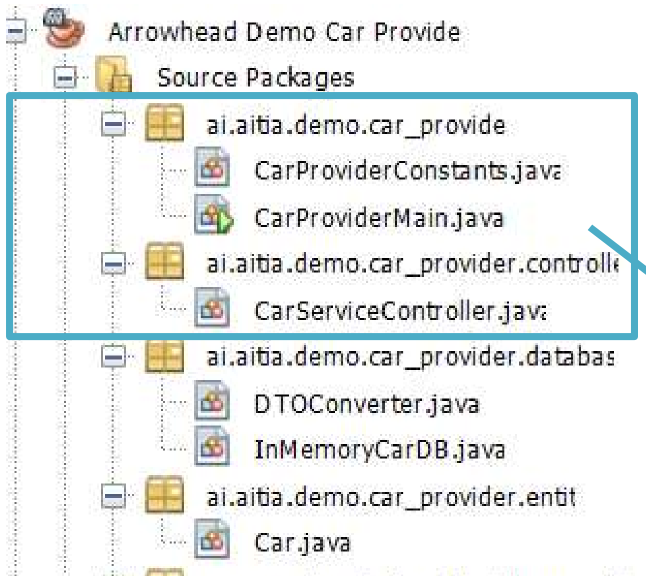
Resources

```
Arrowhead Demo Car Provide
    Source Packages
        ai.aitia.demo.car_provide
            CarProviderConstants.java
```

```java
//-------------------------------------------------------------------------------------------------
@PostMapping(consumes = MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody public CarResponseDTO createCar(@RequestBody final CarRequestDTO dto) {
		if (dto.getBrand() == null || dto.getBrand().isBlank()) {
			throw new BadPayloadException("brand is null or blank");
		}
		if (dto.getColor() == null || dto.getColor().isBlank()) {
			throw new BadPayloadException("color is null or blank");
		}
		final Car car = carDB.create(dto.getBrand(), dto.getColor());
		return DTOConverter.convertCarToCarResponseDTO(car);
}
```

Deploying Your Own Provider - Steps

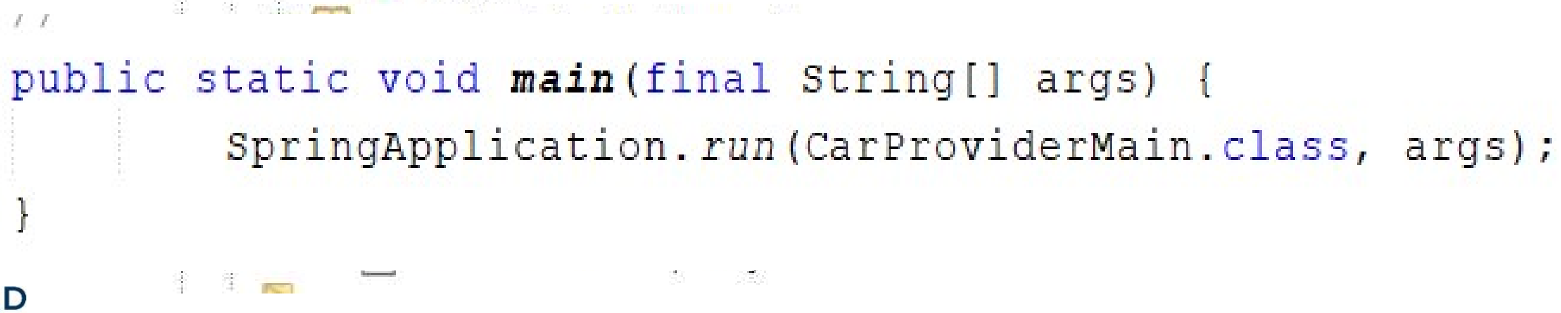
Important functions



```
Arrowhead Demo Car Provide
  Source Packages
    ai.aitia.demo.car_provide
      CarProviderConstants.java
      CarProviderMain.java
    ai.aitia.demo.car_provider.controlle
      CarServiceController.java
    ai.aitia.demo.car_provider.databas
      DTOConverter.java
      InMemoryCarDB.java
    ai.aitia.demo.car_provider.entit
      Car.java
```

Main class

DON'T MODIFY



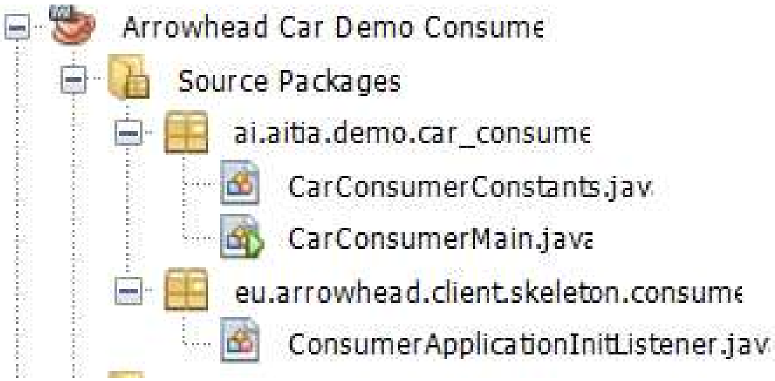
```java
public static void main(final String[] args) {
        SpringApplication.run(CarProviderMain.class, args);
}
```

Deploying Your Own Consumer - Steps

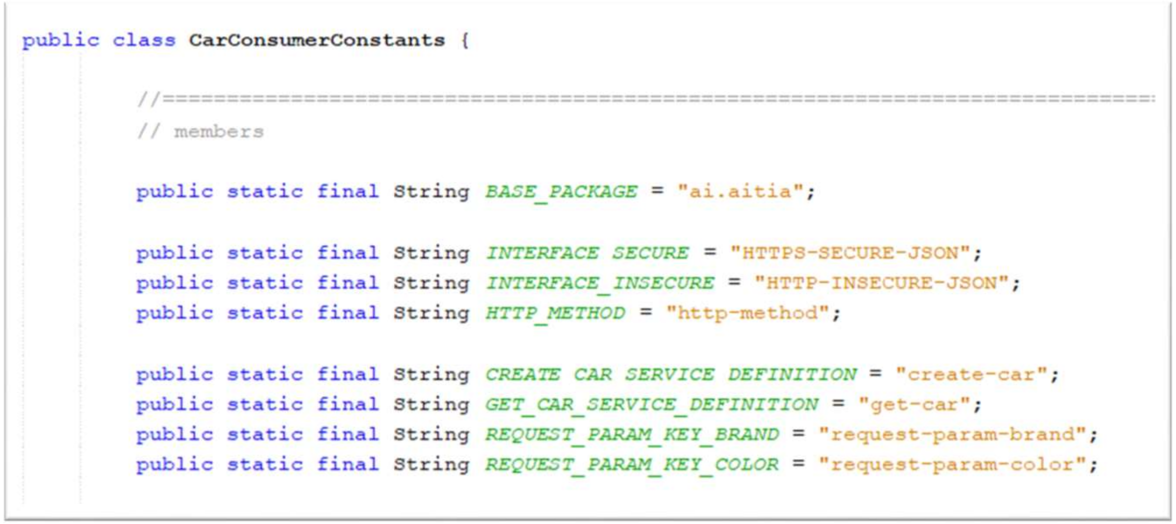
Modify the:

1. Consumer properties and constants.
2. Orchestration Flags.
3. Service Request (Service definition, interfaces and metadata).
4. Implement your own consumer logic (classes, request and response) → Client.

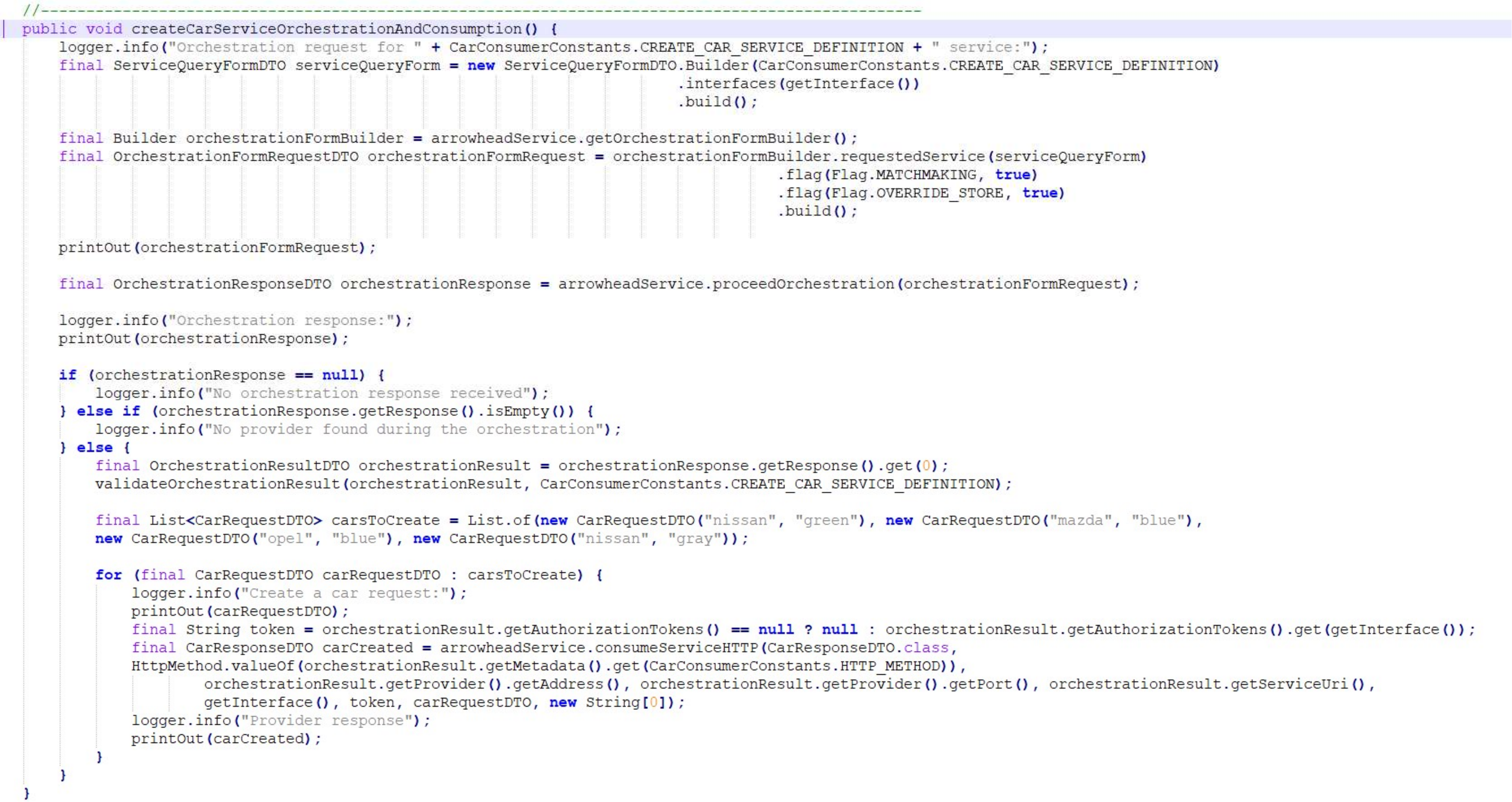
Looking At The Demo Example



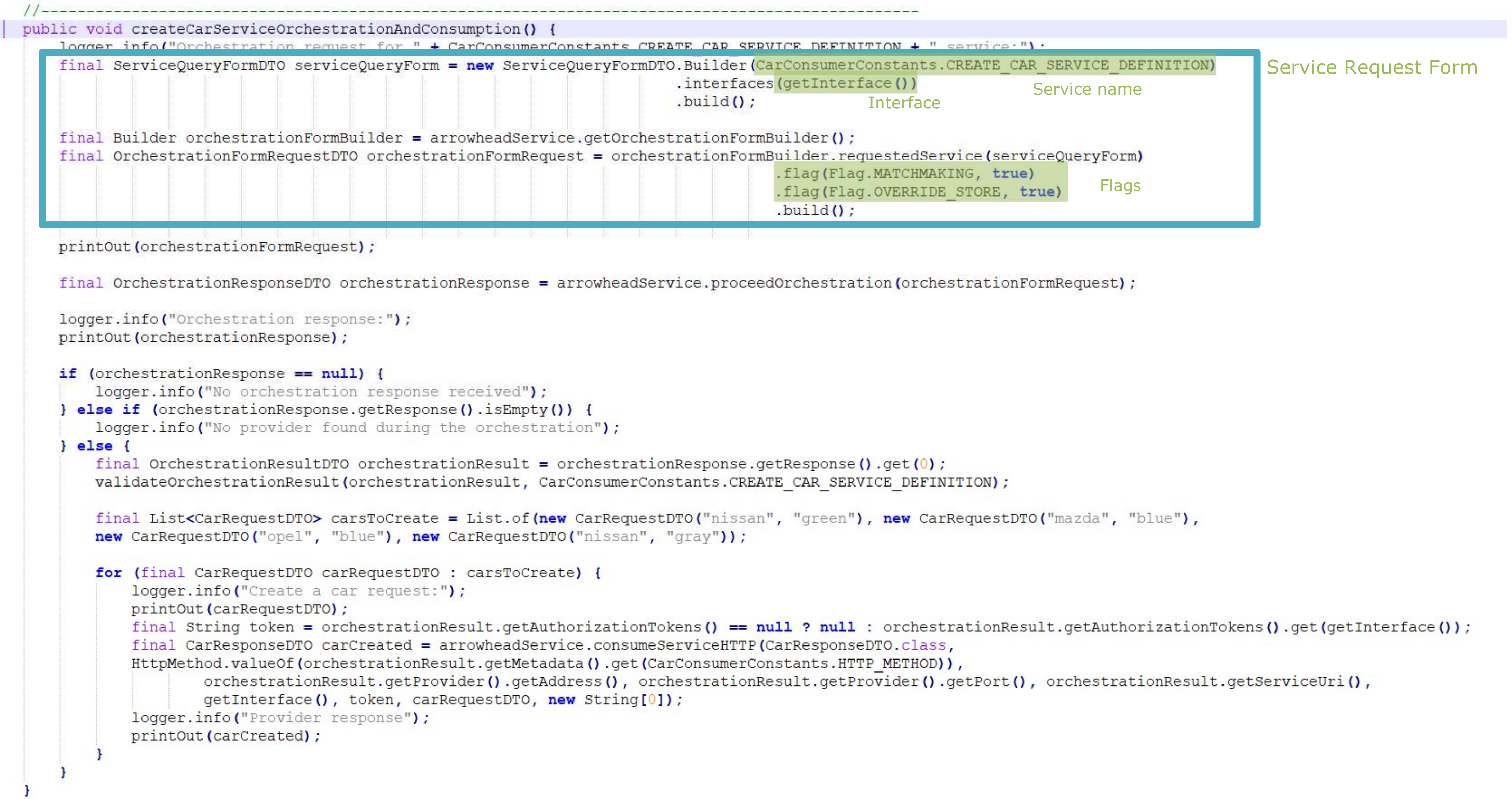
```
Arrowhead Car Demo Consume
  Source Packages
    ai.aitia.demo.car_consume
      CarConsumerConstants.jav
      CarConsumerMain.java
    eu.arrowhead.client.skeleton.consume
      ConsumerApplicationInitListener.jav
```



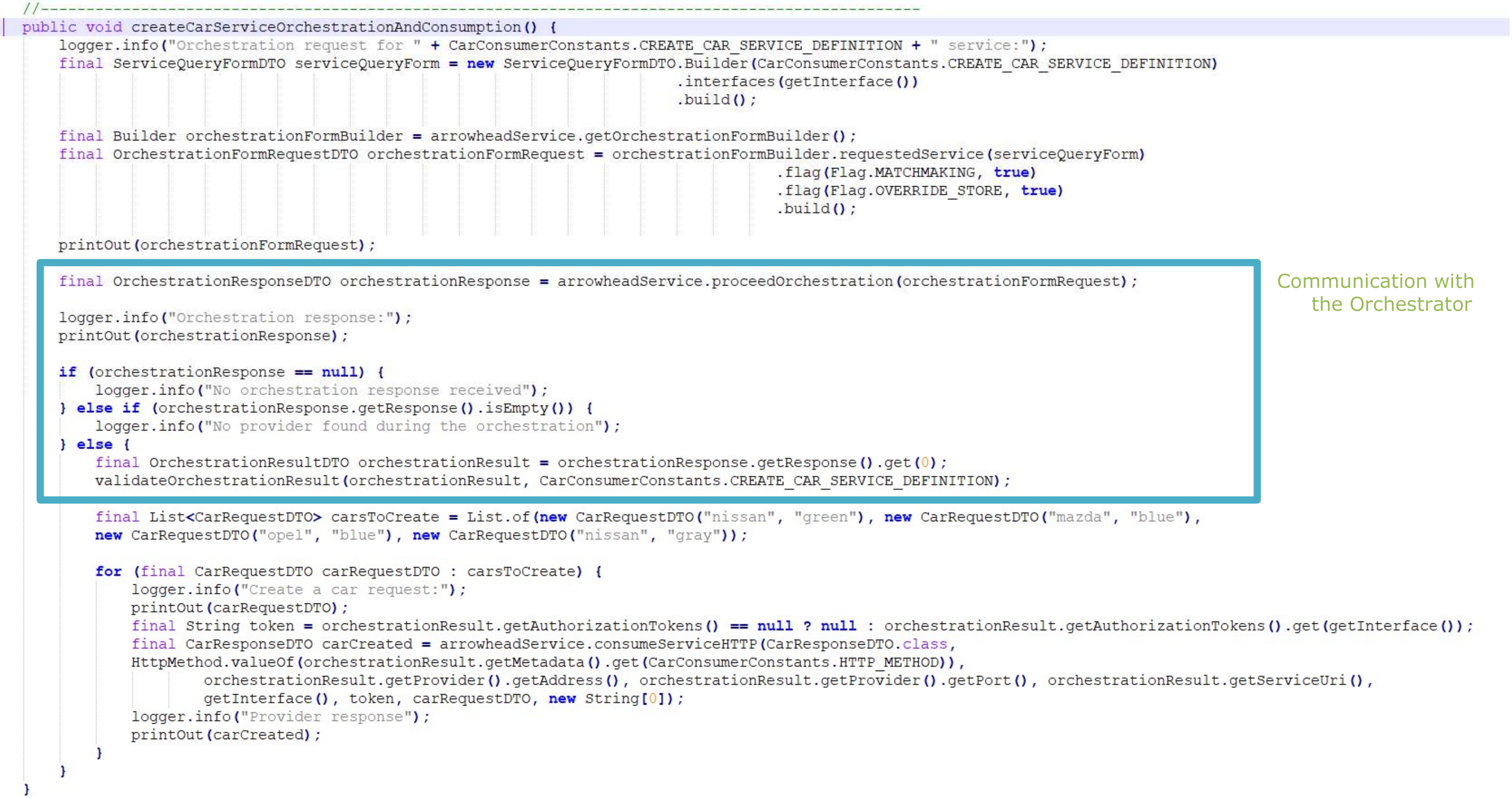
```
public class CarConsumerConstants {

        //=====================================================================================
        // members

        public static final String BASE_PACKAGE = "ai.aitia";

        public static final String INTERFACE SECURE = "HTTPS-SECURE-JSON";
        public static final String INTERFACE_INSECURE = "HTTP-INSECURE-JSON";
        public static final String HTTP_METHOD = "http-method";

        public static final String CREATE CAR SERVICE DEFINITION = "create-car";
        public static final String GET_CAR_SERVICE_DEFINITION = "get-car";
        public static final String REQUEST_PARAM_KEY_BRAND = "request-param-brand";
        public static final String REQUEST_PARAM_KEY_COLOR = "request-param-color";
```



```
//-------------------------------------------------------------------------------------------------
public void createCarServiceOrchestrationAndConsumption() {
    logger.info("Orchestration request for " + CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION + " service:");
    final ServiceQueryFormDTO serviceQueryForm = new ServiceQueryFormDTO.Builder(CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION)
                                                                        .interfaces(getInterface())
                                                                        .build();

    final Builder orchestrationFormBuilder = arrowheadService.getOrchestrationFormBuilder();
    final OrchestrationFormRequestDTO orchestrationFormRequest = orchestrationFormBuilder.requestedService(serviceQueryForm)
                                                                                   .flag(Flag.MATCHMAKING, true)
                                                                                   .flag(Flag.OVERRIDE_STORE, true)
                                                                                   .build();

    printOut(orchestrationFormRequest);

    final OrchestrationResponseDTO orchestrationResponse = arrowheadService.proceedOrchestration(orchestrationFormRequest);

    logger.info("Orchestration response:");
    printOut(orchestrationResponse);

    if (orchestrationResponse == null) {
        logger.info("No orchestration response received");
    } else if (orchestrationResponse.getResponse().isEmpty()) {
        logger.info("No provider found during the orchestration");
    } else {
        final OrchestrationResultDTO orchestrationResult = orchestrationResponse.getResponse().get(0);
        validateOrchestrationResult(orchestrationResult, CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION);

        final List<CarRequestDTO> carsToCreate = List.of(new CarRequestDTO("nissan", "green"), new CarRequestDTO("mazda", "blue"),
        new CarRequestDTO("opel", "blue"), new CarRequestDTO("nissan", "gray"));

        for (final CarRequestDTO carRequestDTO : carsToCreate) {
            logger.info("Create a car request:");
            printOut(carRequestDTO);
            final String token = orchestrationResult.getAuthorizationTokens() == null ? null : orchestrationResult.getAuthorizationTokens().get(getInterface());
            final CarResponseDTO carCreated = arrowheadService.consumeServiceHTTP(CarResponseDTO.class,
            HttpMethod.valueOf(orchestrationResult.getMetadata().get(CarConsumerConstants.HTTP_METHOD)),
                    orchestrationResult.getProvider().getAddress(), orchestrationResult.getProvider().getPort(), orchestrationResult.getServiceUri(),
                    getInterface(), token, carRequestDTO, new String[0]);
            logger.info("Provider response");
            printOut(carCreated);
        }
    }
}
```

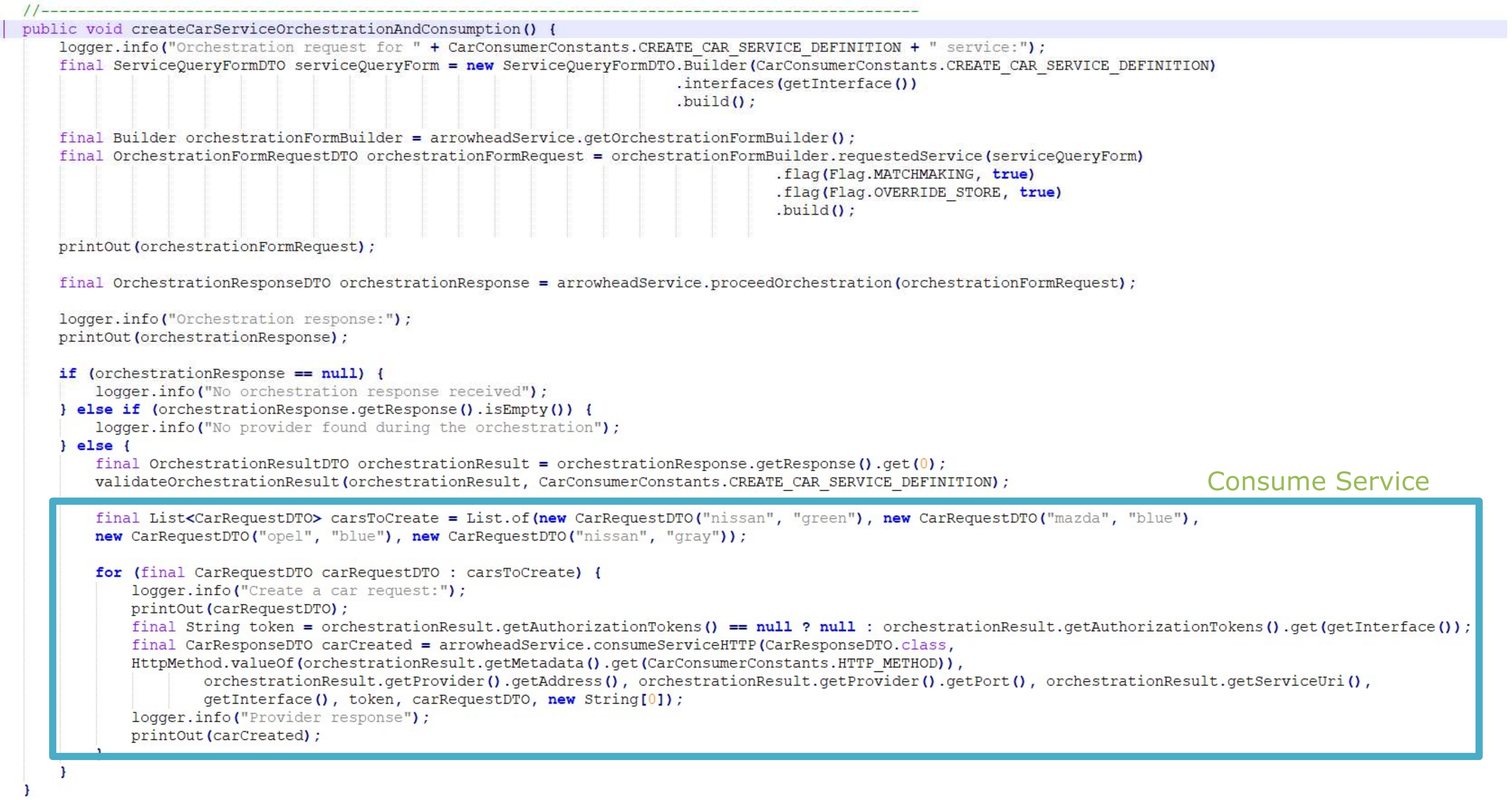


```
//----------------------------------------------------------------------------------------------
public void createCarServiceOrchestrationAndConsumption() {
    logger.info("Orchestration request for " + CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION + " service:");
    final ServiceQueryFormDTO serviceQueryForm = new ServiceQueryFormDTO.Builder(CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION)
                                                                        .interfaces(getInterface())
                                                                        .build();

    final Builder orchestrationFormBuilder = arrowheadService.getOrchestrationFormBuilder();
    final OrchestrationFormRequestDTO orchestrationFormRequest = orchestrationFormBuilder.requestedService(serviceQueryForm)
                                                                                    .flag(Flag.MATCHMAKING, true)
                                                                                    .flag(Flag.OVERRIDE_STORE, true)
                                                                                    .build();

    printOut(orchestrationFormRequest);

    final OrchestrationResponseDTO orchestrationResponse = arrowheadService.proceedOrchestration(orchestrationFormRequest);

    logger.info("Orchestration response:");
    printOut(orchestrationResponse);

    if (orchestrationResponse == null) {
        logger.info("No orchestration response received");
    } else if (orchestrationResponse.getResponse().isEmpty()) {
        logger.info("No provider found during the orchestration");
    } else {
        final OrchestrationResultDTO orchestrationResult = orchestrationResponse.getResponse().get(0);
        validateOrchestrationResult(orchestrationResult, CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION);

        final List<CarRequestDTO> carsToCreate = List.of(new CarRequestDTO("nissan", "green"), new CarRequestDTO("mazda", "blue"),
        new CarRequestDTO("opel", "blue"), new CarRequestDTO("nissan", "gray"));

        for (final CarRequestDTO carRequestDTO : carsToCreate) {
            logger.info("Create a car request:");
            printOut(carRequestDTO);
            final String token = orchestrationResult.getAuthorizationTokens() == null ? null : orchestrationResult.getAuthorizationTokens().get(getInterface());
            final CarResponseDTO carCreated = arrowheadService.consumeServiceHTTP(CarResponseDTO.class,
            HttpMethod.valueOf(orchestrationResult.getMetadata().get(CarConsumerConstants.HTTP_METHOD)),
                    orchestrationResult.getProvider().getAddress(), orchestrationResult.getProvider().getPort(), orchestrationResult.getServiceUri(),
                    getInterface(), token, carRequestDTO, new String[0]);
            logger.info("Provider response");
            printOut(carCreated);
        }
    }
}
```



```
//------------------------------------------------------------------------------------------------
public void createCarServiceOrchestrationAndConsumption() {
    logger.info("Orchestration request for " + CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION + " service:");
    final ServiceQueryFormDTO serviceQueryForm = new ServiceQueryFormDTO.Builder(CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION)
                                                                        .interfaces(getInterface())
                                                                        .build();

    final Builder orchestrationFormBuilder = arrowheadService.getOrchestrationFormBuilder();
    final OrchestrationFormRequestDTO orchestrationFormRequest = orchestrationFormBuilder.requestedService(serviceQueryForm)
                                                                                   .flag(Flag.MATCHMAKING, true)
                                                                                   .flag(Flag.OVERRIDE_STORE, true)
                                                                                   .build();

    printOut(orchestrationFormRequest);

    final OrchestrationResponseDTO orchestrationResponse = arrowheadService.proceedOrchestration(orchestrationFormRequest);

    logger.info("Orchestration response:");
    printOut(orchestrationResponse);

    if (orchestrationResponse == null) {
        logger.info("No orchestration response received");
    } else if (orchestrationResponse.getResponse().isEmpty()) {
        logger.info("No provider found during the orchestration");
    } else {
        final OrchestrationResultDTO orchestrationResult = orchestrationResponse.getResponse().get(0);
        validateOrchestrationResult(orchestrationResult, CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION);

        final List<CarRequestDTO> carsToCreate = List.of(new CarRequestDTO("nissan", "green"), new CarRequestDTO("mazda", "blue"),
        new CarRequestDTO("opel", "blue"), new CarRequestDTO("nissan", "gray"));

        for (final CarRequestDTO carRequestDTO : carsToCreate) {
            logger.info("Create a car request:");
            printOut(carRequestDTO);
            final String token = orchestrationResult.getAuthorizationTokens() == null ? null : orchestrationResult.getAuthorizationTokens().get(getInterface());
            final CarResponseDTO carCreated = arrowheadService.consumeServiceHTTP(CarResponseDTO.class,
            HttpMethod.valueOf(orchestrationResult.getMetadata().get(CarConsumerConstants.HTTP_METHOD)),
                    orchestrationResult.getProvider().getAddress(), orchestrationResult.getProvider().getPort(), orchestrationResult.getServiceUri(),
                    getInterface(), token, carRequestDTO, new String[0]);
            logger.info("Provider response");
            printOut(carCreated);
        }
    }
}
```

Communication with
the Orchestrator



```
//----------------------------------------------------------------------------------------------------
public void createCarServiceOrchestrationAndConsumption() {
    logger.info("Orchestration request for " + CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION + " service:");
    final ServiceQueryFormDTO serviceQueryForm = new ServiceQueryFormDTO.Builder(CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION)
                                                                          .interfaces(getInterface())
                                                                          .build();

    final Builder orchestrationFormBuilder = arrowheadService.getOrchestrationFormBuilder();
    final OrchestrationFormRequestDTO orchestrationFormRequest = orchestrationFormBuilder.requestedService(serviceQueryForm)
                                                                                       .flag(Flag.MATCHMAKING, true)
                                                                                       .flag(Flag.OVERRIDE_STORE, true)
                                                                                       .build();

    printOut(orchestrationFormRequest);

    final OrchestrationResponseDTO orchestrationResponse = arrowheadService.proceedOrchestration(orchestrationFormRequest);

    logger.info("Orchestration response:");
    printOut(orchestrationResponse);

    if (orchestrationResponse == null) {
        logger.info("No orchestration response received");
    } else if (orchestrationResponse.getResponse().isEmpty()) {
        logger.info("No provider found during the orchestration");
    } else {
        final OrchestrationResultDTO orchestrationResult = orchestrationResponse.getResponse().get(0);
        validateOrchestrationResult(orchestrationResult, CarConsumerConstants.CREATE_CAR_SERVICE_DEFINITION);

        final List<CarRequestDTO> carsToCreate = List.of(new CarRequestDTO("nissan", "green"), new CarRequestDTO("mazda", "blue"),
        new CarRequestDTO("opel", "blue"), new CarRequestDTO("nissan", "gray"));

        for (final CarRequestDTO carRequestDTO : carsToCreate) {
            logger.info("Create a car request:");
            printOut(carRequestDTO);
            final String token = orchestrationResult.getAuthorizationTokens() == null ? null : orchestrationResult.getAuthorizationTokens().get(getInterface());
            final CarResponseDTO carCreated = arrowheadService.consumeServiceHTTP(CarResponseDTO.class,
            HttpMethod.valueOf(orchestrationResult.getMetadata().get(CarConsumerConstants.HTTP_METHOD)),
                    orchestrationResult.getProvider().getAddress(), orchestrationResult.getProvider().getPort(), orchestrationResult.getServiceUri(),
                    getInterface(), token, carRequestDTO, new String[0]);
            logger.info("Provider response");
            printOut(carCreated);
        }
    }
}
```

Consume Service

General tips

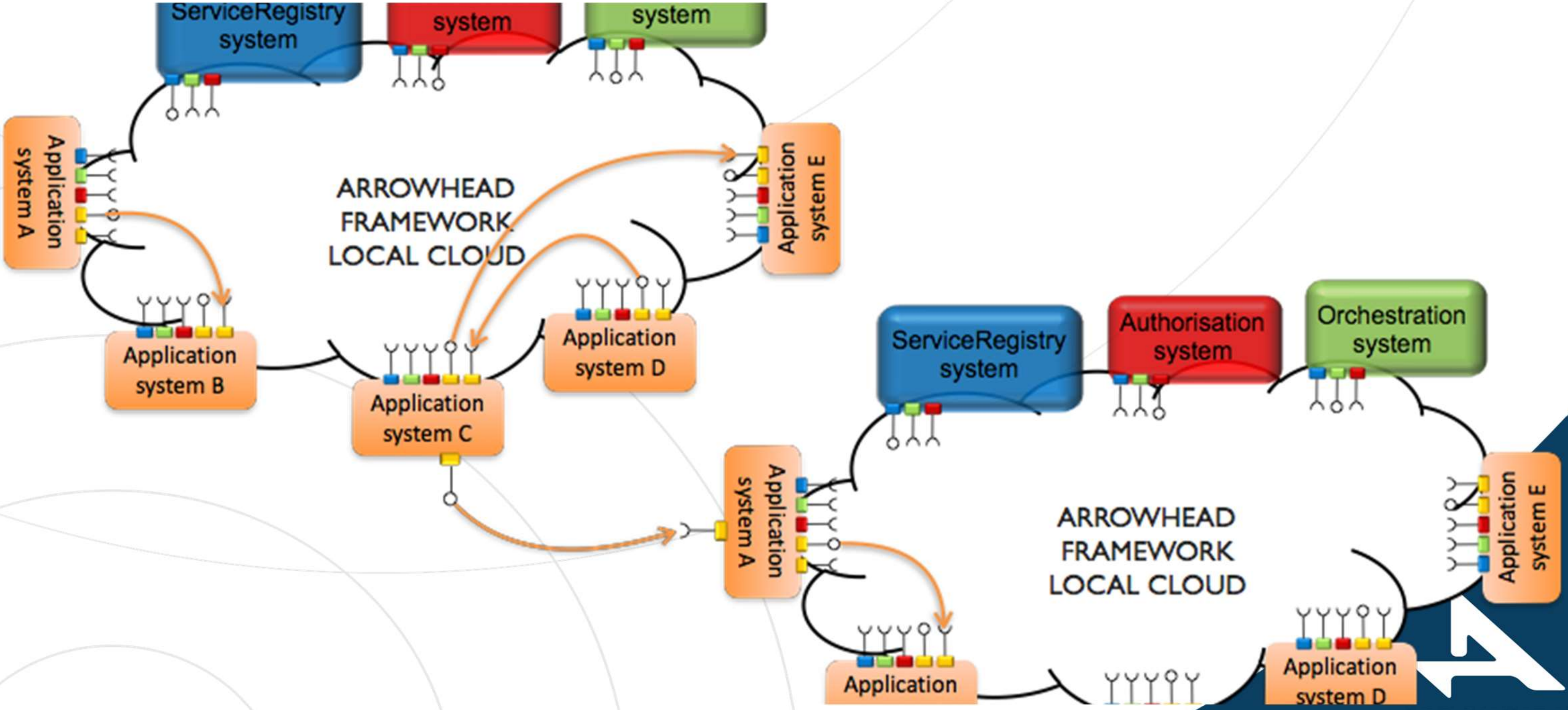
- Read carefully the git-wiki documentation.
- Before creating your own consumer understand what means each flag for the Orchestrator and decide which are the one that you need.
- If you have any problem with the code contact the AITIA team via the GitHub or the Slack group.

READ DOCUMENTATION

Just a little more…

Inter Cloud Communication

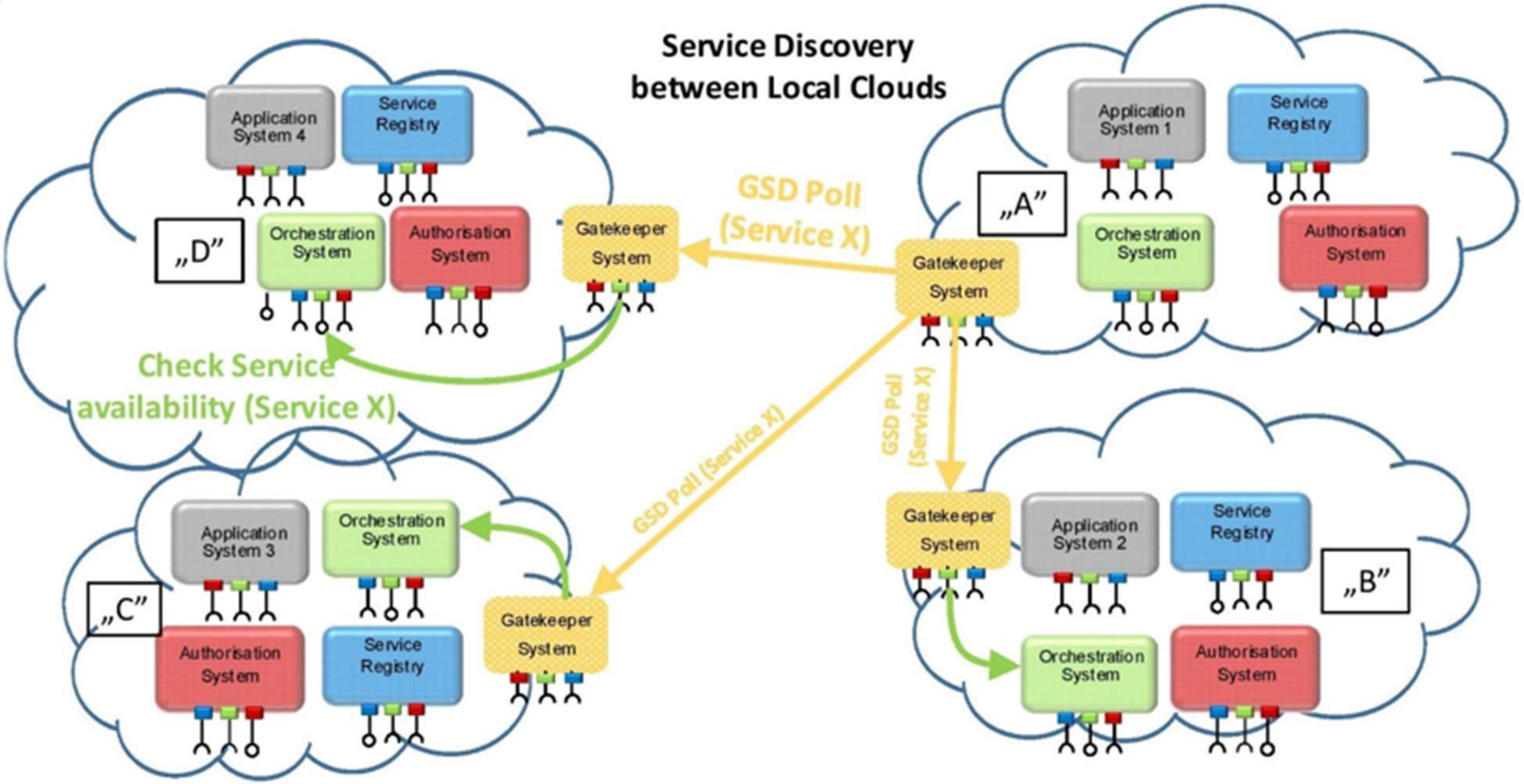
System of Systems



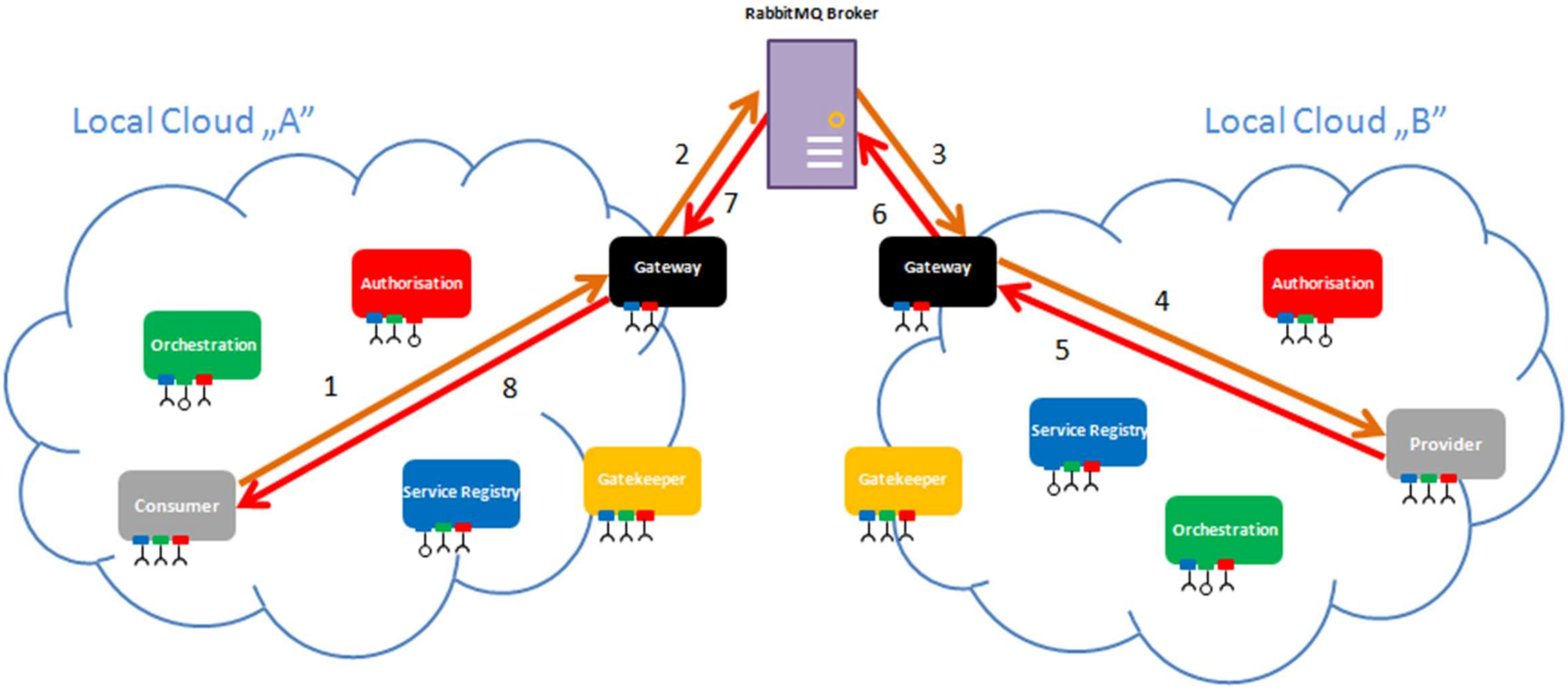
www.arrowhead.eu

ARROWHEAD
fPVN

Local Clouds Interaction

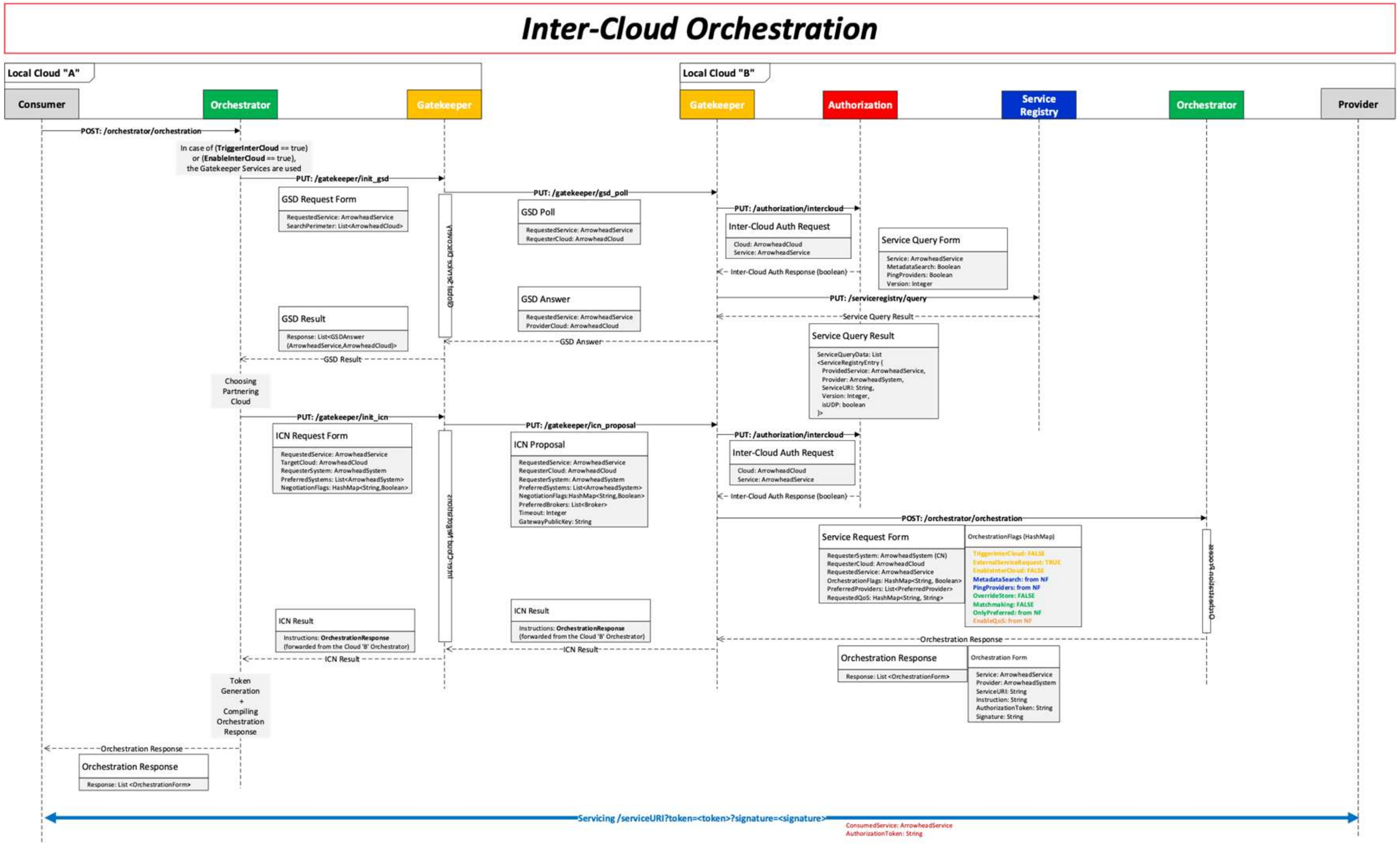


Gatekeeper - Gateway



www.arrowhead.eu

Gatekeeper Operation


Inter-Cloud Orchestration
Local Cloud "A"
Consumer
Orchestrator
Gatekeeper
Local Cloud "B"
Gatekeeper
Authorization
Service Registry
Orchestrator
Provider
POST: /orchestrator/orchestration
In case of (TriggerInterCloud == true)
or (EnableInterCloud == true),
the Gatekeeper Services are used
PUT: /gatekeeper/init_gsd
Global Service Discovery
PUT: /gatekeeper/gsd_poll
PUT: /authorization/intercloud
Inter-Cloud Auth Response (boolean)
PUT: /serviceregistry/query
Service Query Result
GSD Answer
GSD Result
Choosing Partnering Cloud
PUT: /gatekeeper/init_icn
Inter-Cloud Negotiations
PUT: /gatekeeper/icn_proposal
PUT: /authorization/intercloud
Inter-Cloud Auth Response (boolean)
POST: /orchestrator/orchestration
Orchestration Process
Orchestration Response
ICN Result
ICN Result
Token Generation + Compiling Orchestration Response
Orchestration Response
Servicing /serviceURI?token=<token>?signature=<signature>

GSD Request Form
RequestedService: ArrowheadService
SearchPerimeter: List<ArrowheadCloud>

GSD Poll
RequestedService: ArrowheadService
RequesterCloud: ArrowheadCloud

Inter-Cloud Auth Request
Cloud: ArrowheadCloud
Service: ArrowheadService

Service Query Form
Service: ArrowheadService
MetadataSearch: Boolean
PingProviders: Boolean
Version: Integer

GSD Answer
RequestedService: ArrowheadService
ProviderCloud: ArrowheadCloud

GSD Result
Response: List<GSDAnswer
(ArrowheadService,ArrowheadCloud)>

Service Query Result
```
ServiceQueryData: List
<ServiceRegistryEntry {
  ProvidedService: ArrowheadService,
  Provider: ArrowheadSystem,
  ServiceURI: String,
  Version: Integer,
  isUDP: boolean
}>
```

ICN Request Form
RequestedService: ArrowheadService
TargetCloud: ArrowheadCloud
RequesterSystem: ArrowheadSystem
PreferredSystems: List<ArrowheadSystem>
NegotiationFlags: HashMap<String,Boolean>

ICN Proposal
RequestedService: ArrowheadService
RequesterCloud: ArrowheadCloud
RequesterSystem: ArrowheadSystem
PreferredSystems: List<ArrowheadSystem>
NegotiationFlags:HashMap<String,Boolean>
PreferredBrokers: List<Broker>
Timeout: Integer
GatewayPublicKey: String

Inter-Cloud Auth Request
Cloud: ArrowheadCloud
Service: ArrowheadService

Service Request Form
RequesterSystem: ArrowheadSystem (CN)
RequesterCloud: ArrowheadCloud
RequestedService: ArrowheadService
OrchestrationFlags: HashMap<String, Boolean>
PreferredProviders: List<PreferredProvider>
RequestedQoS: HashMap<String, String>

OrchestrationFlags (HashMap)
TriggerInterCloud: FALSE
ExternalServiceRequest: TRUE
EnableInterCloud: FALSE
MetadataSearch: from NF
PingProviders: from NF
OverrideStore: FALSE
Matchmaking: FALSE
OnlyPreferred: from NF
EnableQoS: from NF

ICN Result
Instructions: **OrchestrationResponse**
(forwarded from the Cloud 'B' Orchestrator)

ICN Result
Instructions: **OrchestrationResponse**
(forwarded from the Cloud 'B' Orchestrator)

Orchestration Response
Response: List <OrchestrationForm>

Orchestration Form
Service: ArrowheadService
Provider: ArrowheadSystem
ServiceURI: String
Instruction: String
AuthorizationToken: String
Signature: String

Orchestration Response
Response: List <OrchestrationForm>

ConsumedService: ArrowheadService
AuthorizationToken: String

Q&A

This research work has been funded by the European Commission, through the European H2020 research and innovation programme, ECSEL Joint Undertaking, and National Funding Authorities from 18 involved countries under the research project Arrowhead Tools with Grant Agreement no. 826452.

ARROWHEAD
TOOLS