Table of Contents

Kernel Platform Guides

The Kernel Platform is a data platform that normalizes access to the company's data and its telco capabilities.

These guides are intended to be read by technical people, and specifically release engineers, operators and administrators that are willing to:

- Install the Kernel Platform.
- Operate and monitor the Kernel Platform.
- Administer the Kernel Platform.

The release notes are also avalable in this document.

The information contained hereinafter is deemed as confidential. In no case this document should be disclosed to any third parties. If you miss something or something is not clear enough, please let us know at the Kernel Platform Community Site so we can continuously improve these guides.

Kernel Platform Deployment Guide

This guide describes how to install the Kernel Platform.

The procedure is valid to install the Kernel Platform from scratch, and also to apply a patch to an existing platform (also known as an in-place upgrade).

⚠️ in-place upgrades are only supported for patches. If you need to upgrade to a new release with major or minor changes (for example, 3.y.z to 4.y.z or 3.9.z to 3.10.z), check the "parallel upgrade" section.

Prerequisites

The installer is compatible with Linux and MacOS and can be run from your computer or from a dedicated virtual machine on the cloud. Most part of the deployment runs in a Docker container to guarantee that it behaves consistently on any system. The only requirements are **Python 3.10+, Docker v17.06+** and the tool `curl`.

In **Linux** (Debian-based), you can install them with the following commands:

```
$ sudo apt update && sudo apt install -y curl python3
$ curl -sSL https://get.docker.com | sh
```

In **MacOS**, download and install Docker for Mac v17.06+. You can use brew to install Python if needed.

Azure cloud setup

The Kernel Platform can be deployed on Microsoft Azure cloud (AWS support is dropped since release/4.2). You need:

- A **valid Azure subscription**.
- A Service Principal previously provisioned in the Azure Active Directory (Azure AD). You will be asked for client_id / client_secret.

⚠️ Use a separate subscription for the Kernel Platform. Do not reuse an existing one to avoid throttling issues.

You must ensure the following permissions in the resource creation:

- The "Application administrator" role in Active Directory.
- The following roles assigned in the Azure subscription (Access Control / IAM):
  - Contributor
  - Cloud application administrator
  - Key Vault Crypto Officer
  - App Configuration Data Owner
  - Key Vault Administrator
- A custom role previously created called Installer with the following permissions:
  - Microsoft.Authorization/policyAssignments/write
  - Microsoft.Authorization/policyAssignments/delete
  - Microsoft.Authorization/roleAssignments/write
  - Microsoft.Authorization/roleAssignments/delete
  - Microsoft.Authorization/locks/*
- The "ConfigService.Admin" Configuration Service app role

Once the Azure AD User is configured, you need to export the following environment variables with the username, password and subscription identifier you want to use:

```
$ export AZURE_TENANT_ID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx
$ export AZURE_SUBSCRIPTION_ID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx
$ export AZURE_CLIENT_ID=xxx
$ export AZURE_CLIENT_SECRET=xxx
```

Vault password

The installer reads some configurations that are considered secrets, so they are stored in Vaults, which are encrypted files or encrypted values inside a file. In order for the installer to decrypt those Vaults you have to export an environment variable named `VAULT_PASSWORD` containing the Vault password.

```
$ export VAULT_PASSWORD=xxx
```

Installer

The Kernel Platform installation is done with two installers: one to deploy and configure the Kernel Platform State, and another one to deploy and configure the rest of the required infrastructure and services. The installers are stored in a S3 bucket that belongs to the Kernel Platform Team. To download them, export the **user credentials provided by the Kernel Platform Support team**:

```
$ export AWS_ACCESS_KEY_ID=xxx
$ export AWS_SECRET_ACCESS_KEY=xxx
```

Then list the available installers and download the appropriate version (requires AWS CLI):

```
$ aws s3 ls s3://4p-installers/<ob>/fourth-platform-state
$ aws s3 cp s3://4p-installers/<ob>/fourth-platform-state/<version>/fourth-platform-state-<ob>-<version>.tgz .
$ aws s3 ls s3://4p-installers/<ob>/fourth-platform
$ aws s3 cp s3://4p-installers/<ob>/fourth-platform/<version>/fourth-platform-<ob>-<version>.tgz .
$ aws s3 ls s3://4p-installers/<ob>/config-service
$ aws s3 cp s3://4p-installers/<ob>/config-service/<version>/config-service-<ob>-<version>.tgz .
```

ℹ️ Please contact the Kernel Platform Support Team if you have any issue accessing or using the installers.

Configuration

The Kernel Platform configuration is stored in the Configuration Service, which can be manged using the `baikops` tool. The installers read the configuration from such a service to deploy the platform.

It is important to keep the configuration up to date with the real Kernel Platform configuration. Otherwise, some manual changes you might have done on the platform could be lost in the face of a new deployment.

Configuration Service

The Configuration Service is a service in charge of storing configurations used later to deploy the Kernel Platform and the Kernel Platform State. Configurations stored in the Configuration Service replace the config files used so far to deploy the Kernel Platform. This way, configurations stored in the Configuration Service become the source of truth, since the installer will read configs from such a service to deploy the platform.

The first time this service is going to be used, the configuration will be uploaded using the `baikops` tool from the config files included in the installer, and this will be the last time the installer will include config files for the Kernel Platform.

Configurations in the Configuration Service are identified by the phase of the installer they belong to (`state` , `infra` , `core` or `tenant`) and a label which identifies the deployment (e.g. `state=es-pro` for the state or `infra=es-pro,core=es-pro` for the core).

Config files

The first time the Configuration Service is used, configurations must be uploaded to the service from the config files included in the installer under the `config` directory by using the `baikops` tool. The State installer comes with some configs under the `config/state` directory while the other installer comes with some configs under the `config/infra` , `config/core` and `config/tenant` directories. For example:

```
$ ls -l config/state

-rw-r--r-- 1 staff staff  873 jun 21 11:43 es-pre.override.yml
-rw-r--r-- 1 staff staff  774 jun 21 11:43 es-pre.yml
-rw-r--r-- 1 staff staff  873 jun 21 11:43 es-pro.override.yml
-rw-r--r-- 1 staff staff  778 jun 21 11:43 es-pro.yml
```

```
$ ls -l config/tenant

-rw-r--r-- 1 staff staff  2104 jun 22 12:49 es-pre-o2.override.yml
-rw-r--r-- 1 staff staff  1107 jun 22 12:49 es-pre-o2.yml
-rw-r--r-- 1 staff staff 41762 jun 22 12:49 es-pre.override.yml
-rw-r--r-- 1 staff staff  5865 jun 22 12:49 es-pre.yml
-rw-r--r-- 1 staff staff  2104 jun 22 12:49 es-pro-o2.override.yml
-rw-r--r-- 1 staff staff  1107 jun 22 12:49 es-pro-o2.yml
-rw-r--r-- 1 staff staff  5474 jun 22 12:49 es-pro.override.yml
-rw-r--r-- 1 staff staff  4759 jun 22 12:49 es-pro.yml
```

Under `config/tenant` two kind of configs can be found: the ones used by the default tenant (deployed along with the core) having the same names as the core configs, and other configs used to deploy additional tenants.

Vault files

As stated before, from Kernel Platform release 5.0 onwards the config files are modified, and `vault.yml` file is no longer used. As a replacement, from Kernel Platform release code name Ontario onwards, the cluster installer comes with several vaulted files:

- `config/infra/<ob>-pre.override.yml`
- `config/infra/<ob>-pro.override.yml`
- `config/core/<ob>-pre.override.yml`
- `config/core/<ob>-pro.override.yml`
- `config/tenant/<ob>-pre-<tenant>.override.yml`
- `config/tenant/<ob>-pro-<tenant>.override.yml`

and for the state installer:

- `config/state/<ob>-pre.override.yml`

- `config/state/<ob>-pro.override.yml`

that allows to configure the secrets and passwords needed to assurance the services deployed in Kernel Platform. These files are encrypted with the same password as the zip file of the installer and must be re-encrypted with the same installer's password once all the values are filled-out.

⚠️ It is not possible to rotate passwords between upgrades. The services' password from `CLUSTER_FROM` and `CLUSTER_TO` must match, except for the new services' password that has to be included.

The same steps must be done also for configs stored in `config/tenant`, if there are more tenants.

Validate configurations

It is recommended to validate that everything looks good before installing the Kernel Platform. The Kernel Platform installer allows to validate the following before actually installing the platform:

- The Azure credentials are correct.
- The Azure user has the proper roles assigned.
- The configuration is correct.

The following commands validate the main phases of the installation process:

```
$ ./baikal-state validate --state $STATE
$ ./baikal infra validate --state $STATE --infra $INFRA
$ ./baikal core validate --state $STATE --infra $INFRA --core $CORE
$ ./baikal tenant validate --state $STATE --infra $INFRA --core $CORE --tenant $TENANT
```

Those commands do not create or modify any Kernel Platform resource, simply validate that a minimal set of requirements are fulfilled to install the Kernel Platform.

Note that these commands validate the configurations actually stored in the Configuration Service. For them to work, the Config Service must have been previously installed, and the configurations must have been pushed as described below.

ℹ️ For more information on creating Azure users, see the complete process to add new users to Azure AD in the Microsoft Azure documentation.

ℹ️ Before deploying the Kernel Platform, check the limits associated to your subscription in the Azure Portal. Go to the "Subscriptions" blade, select the subscription and click on "Usage + Quotas". The number and type of instances needed is dynamic and depends on how you configure your deployment config, as will be explained later.

Deployment

The installation process is divided into the following steps:

- From the Configuration Service installer:
    1. This step creates the Configuration Service if it does not exist and deploy API for this service. This step configures the Azure subscription and creates a set of cloud resources that prepares the subscription to hold the Kernel Platform Configuration Service.
- From the State installer:
    1. Push the State configuration to the Configuration Service, in order to be used during the State deployment.
    2. Create the Kernel Platform State, that is, a set of cloud services where the Kernel Platform stores data that must be persisted even in the case of uninstalling it or upgrading it. It contains databases, a logging subsystem, and storage accounts where important assets are stored such as datasets and algorithms. This step configures the Azure subscription

and creates a set of cloud resources that prepares the subscription to hold the Kernel Platform State.

- From the Kernel Platform installer:

  1. Push the infra, core and default tenant configurations to the Configuration Service, in order to be used during the deployment.
  2. Create the infrastructure where the services will be deployed (Kubernetes cluster, Application GW, etc.) along with a subset of the Kernel Platform services (system services) that provide operational features such as logging, alerting, metrics, load balancing, etc. This step configures the Azure subscription and creates a set of cloud resources that prepares the subscription to hold the Kernel Platform Configuration Service.
  3. Deploy the Kernel Platform core services on top of the infrastructure. This step creates the default tenant.
  4. Optionally, create secondary tenants, which configures the core services to work with those tenants. Remember to push the secondary tenant configurations before creating them.

State and infrastructure resources are defined as code and managed by Terraform.

⚠️ Do not try to manage the Kernel Platform Terraform state on your own. You should never manually create, modify or remove cloud resources, either. Otherwise the platform status won't be in sync with the state managed by Terraform, causing problems.

⚠️ Since release/4.2 it is possible reuse a set of IP addresses for the Outbound rule in the cluster. You might need to whitelist these IP addresses before deploying the platform, to allow access to/from external services. Please go to the section "Outbound IP Addresses Preprovision" for additional information.

Configuration Service installer

The following steps will be performed from the config-service installer.

Configuration Service

```
# TAG: Tag used to identify this Telefónica Kernel Config Service instance for observability purposes.
# REGION: Azure region where Telefónica Kernel Config Service will be installed.
./config-service install --tag <TAG> --region <REGION> --yes
```

State installer

The following steps will be performed from the State installer.

Push configuration

The Kernel Platform State configuration must be uploaded to the Configuration Service only if it has not been done before. Once the configuration is in the Configuration Service, it will become the source of truth.

```
$ ./baikal-state config push --state $STATE --config $CONFIG
```

State

To create the **state**:

```
$ ./baikal-state create --state $STATE
```

The whole process takes around 30 minutes. When it finishes, a deployment summary is shown.

```
State creation summary:
  Access to the Kubernetes cluster:
      export KUBECONFIG=/tmp/baikal/delivery/output/state-create-es-pre/kubernetes/kubeconfig.yml
    or:
      az aks get-credentials --name baikal-state-es-pre --resource-group baikal-es-pre-state-rg --subscription
ab6cc283-033c-422c-9285-7812eef487c1
```

Kernel Platform installer

The following steps will be performed from the Kernel Platform installer.

Push configurations

The Kernel Platform configurations must be uploaded to the Configuration Service only if it has not been done before. Once the configurations are in the Configuration Service, they will become the source of truth.

```
$ ./baikal infra config push --infra $INFRA --config $CONFIG
$ ./baikal core config push --infra $INFRA --core $CORE --config $CONFIG
$ ./baikal tenant config push --infra $INFRA --tenant $CORE --config $CONFIG
```

Note that the default tenant configuration must be pushed independently from the core configuration and its name is the same as the core.

Infrastructure

To create the **infrastructure**:

```
$ ./baikal infra create --state $STATE --infra $INFRA
```

The whole process takes around 10 minutes. When it finishes, a deployment summary is shown.

```
Infrastructure creation summary:
  - Access to the Kubernetes cluster:
      export KUBECONFIG=/tmp/baikal/delivery/output/infra-create-baikal-es-pre/kubernetes/kubeconfig.yml
    or:
      az aks get-credentials --name es-pre --resource-group baikal-es-pre-cluster-rg --subscription ab6cc283-03
3c-422c-9285-7812eef487c1

The Kernel Platform infrastructure creation has successfully finished.
```

ℹ️ Some cloud load balancers are created at this point. Every time the infrastructure services are redeployed, their IP addresses can change. In Azure, both the ingress IP addresses and the egress IP addresses (those seen by third party services that are accessed from the Kernel Platform) can change. See more details about the Kernel Platform networking details at the cloud infrastructure section of the operations guide.

Core services

To create the **core services**:

```
$ ./baikal core create --state $STATE --infra $INFRA --core $CORE
```

The process should take around 90 minutes. It also shows a deployment summary when it finishes, with links to the most relevant services.

```
Core creation summary:
  Fourth Platform services:
    https://alerts.es-pre.baikalplatform.com
    https://auth.es-pre.baikalplatform.com
    https://api.es-pre.baikalplatform.com
    https://rawlogs.es-pre.baikalplatform.com
    https://dashboard.es-pre.baikalplatform.com
    https://logs.es-pre.baikalplatform.com
    https://www.es-pre.baikalplatform.com
    https://metrics.es-pre.baikalplatform.com
    https://spark.es-pre.baikalplatform.com

  Access to the Kubernetes cluster:
      export KUBECONFIG=/tmp/baikal/delivery/output/core-create-es-pre-es-pre/kubernetes/kubeconfig.yml
    or:
      az aks get-credentials --name es-pre --resource-group baikal-es-pre-cluster-rg --subscription ab6cc283-03
3c-422c-9285-7812eef487c1

The Kernel Platform core resources and the default tenant have been successfully created.
```

The installation is fully automated and designed to be idempotent. This means that, if something goes wrong, it is safe to run the installer again. At the end of the deployment process, you should verify that the installation has been successful.

⚠️ Once the Kernel Platform deployment finishes, you will have a couple of kubeconfig files that grants full access to manage the State and Kernel Platform clusters. **Distributing the "root" kubeconfig file it is STRONGLY DISCOURAGED in production environments**.

For security reasons, nobody should use the "root" kubernetes files and they should never be distributed. Instead, a different kubeconfig file must be generated for each user that needs to access the cluster (more info in the Operations Guide - RBAC Security Section). This way:

- Each user can use the role that best matches each use case (view, edit, admin).
- Accesses can be revoked anytime.
- Operations on the Kubernetes cluster can be logged and audited.

Note that the core creation implies the creation of the default tenant which is named after the core. Secondary tenants can be created later.

Tenant

⚠️ New tenant callbacks must be registered in the core O365 application used by the portal backend in authentication. If it is possible, this request should be managed through ARGONAUTA. If you don't have access, it should be managed through the global operation or development teams. For the update request you will need indicate the client_id of the application (you can find it in the portal-backend secret as `PORTAL_OIDC_CLIENTID`) and the new URL callback for the additional tenant: https://www.$TENANT.baikalplatform.com/oauth/oidc/callback

To create a secondary **tenant**:

First of all, push the tenant/s configuration to the Configuration Service. You must push the configuration for each tenant you want to create:

```
$ ./baikal tenant config push --infra $INFRA --tenant $TENANT --config $CONFIG
```

Then, create the tenant/s:

```
$ ./baikal tenant create --state $STATE --infra $INFRA --core $CORE --tenant $TENANT1,$TENANT2,...,$TENANTN
```

The process should take around 60 minutes. It also shows a deployment summary when it finishes, with links to the most relevant services.

```
Tenant creation summary:
  Telefónica Kernel services:
    https://auth.es-pre-o2.baikalplatform.com
    https://api.es-pre-o2.baikalplatform.com
    https://www.es-pre-o2.baikalplatform.com
    https://spark.es-pre-o2.baikalplatform.com

  Access to the Kubernetes cluster:
      export KUBECONFIG=/tmp/baikal/delivery/output/tenant-create-es-pre-es-pre-o2/kubernetes/kubeconfig.yml
    or:
      az aks get-credentials --name es-pre --resource-group baikal-es-pre-cluster-rg --subscription ab6cc283-03
3c-422c-9285-7812eef487c1

The Kernel Platform tenant resources have been successfully created.
```

During the creation of a secondary tenant, core services could be restarted, but the service will not be disrupted as long as there are more than 1 replica of each service.

Resource groups

Several resource groups are created to organize the different cloud assets:

1. `baikal-subscription-rg` . It is shared for all the Kernel platforms in the same Azure subscription. It contains the Terraform state and the reserved IP prefixes.
2. `baikal-backups-rg` . It contains storage accounts where the Kernel Platform stores the backups that are periodically performed to allow the platform recovery in case of a disaster.
3. `baikal-${STATE}-state-rg` . It contains all the cloud assets that belong to the Kernel Platform state.
4. `baikal-${STATE}-aks-state-rg` . It contains the resources that conform the state Kubernetes cluster, and it si entirely managed
5. `baikal-${CLUSTER}-cluster-rg` . It contains the Kernel Platform infrastructure (Kubernetes cluster, Application GW, etc.).
6. `baikal-${CLUSTER}-aks-rg` . It contains the resources that conform the Kubernetes cluster, and it si entirely managed by Azure AKS.
7. `baikal-${CLUSTER}-cluster-${CORE}-core-rg` . It contains several resources that are dynamically created by the Kernel Platform when it is running.

Undeployment

In case you need to uninstall the Kernel Platform, the resources that were created in the installation process can be removed.

⚠️ This action is permanent and can not be undone. If you just want to temporarily stop an existing platform, take a look at the advanced options. Nevertheless, the backups are never removed.

Kernel Platform installer

The following steps will be performed from the Kernel Platform installer. Note that configurations must be in the Configuration Service in order to uninstall the platform.

Tenant

To delete a **tenant**:

```
$ ./baikal tenant delete --infra $INFRA --core $CORE --tenant $TENANT
```

Core services

To delete the **core services**:

```
$ ./baikal core delete --state $STATE --infra $INFRA --core $CORE
```

This will also delete all the tenants configured in that core.

Infrastructure

Remember to remove the services first. If you remove the infrastructure without removing the services, some cloud resources (DNS entries, storage accounts) won't be deleted.

Once the services have been deleted, you can delete the **infrastructure**:

```
$ ./baikal infra delete --state $STATE --infra $INFRA
```

⚠️ All the infrastructure resources are removed but data stored in the state is kept in the `baikal-$STATE-state-rg` resource group. You can always create a new infrastructure using the same state or, if you want to remove also the state, go to the following section.

ℹ️ If you plan to uninstall and install new Kernel Platform environments frequently (e.g. weekly), remember to use staging certificates. It is important because otherwise you will reach the weekly limit of Let's Encrypt certificates. To do so, set the `letsencrypt_staging` variable in the deployment config to `True`.

```
letsencrypt_staging: True
```

Delete configurations

In case you also want to delete the configurations from the Configuration Service:

```
$ ./baikal infra config delete --infra $INFRA
$ ./baikal core config delete --infra $INFRA --core $CORE
$ ./baikal tenant config delete --infra $INFRA --tenant $CORE
```

Delete also secondary tenant configurations if needed:

```
$ ./baikal tenant config delete --infra $INFRA --tenant $TENANT
```

State installer

The following steps will be performed from the State installer.

State

To delete the **state**:

```
$ ./baikal-state delete --state $STATE
```

Note that the configuration must be in the Configuration Service in order to delete the state.

⚠️ Data stored in the state will be removed forever. Nevertheless, backups are kept in the `baikal-backups-rg` resource group. If you don't ever need those backups, remove them manually.

Delete configuration

In case you also want to delete the configuration from the Configuration Service:

```
$ ./baikal-state config delete --state $STATE
```

The deployment guide describes how to install the Kernel Platform and the basic commands to create the state, the infrastructure, and deploy core services, as well as additional tenants.

Moreover, there are additional actions you can execute, such as stopping and starting a platform.

Alternative FQDNs

When the Kernel Platform is deployed, services are exposed to the Internet through FQDNs that looks like `http://auth.<TENANT>.baikalplatform.com` and are protected using TLS with a certificate issued by Let's Encrypt. Though such FQDNs are fully functional, they are not using the OB-branded domain name, and this may be considered an UX issue depending on how much exposed to the OB customers each service is (e.g. the Authserver uses to be more exposed to the OB customers than the dashboards).

From Kernel Platform release 5.0 onwards, it is possible to configure alternative FQDNs for exposing each service. Those alternative FQDNs are fully managed by the OB, and thus it can be an OB-branded FQDN such as `login.4p.movistar.es`.

ℹ️ Not all Kernel Platform services support an alternative FQDN. Please, ask the Kernel Platform Team in case you want to use this feature.

Prerequisites

The following resources must be managed by the OB before configuring an alternative FQDN:

1. An OB-branded FQDN must be setup in the OB's DNS service with a `CNAME` record (e.g. `login.4p.movistar.es`) pointing to the default FQDN (e.g. `auth.<TENANT>.baikalplatform.com`).
2. Since Kernel Platform services are protected by TLS, a valid certificate must be provided by the OB whose CN matches the OB-branded FQDN. Both the private key and the public certificate are needed to configure the Kernel Platform.

Configuration

The alternative FQDN must be configured in the tenant config before creating a tenant:

1. Create a new entry in the tenant config under `dns.<service>` named `alternative_fqdns` and then create an entry with the FQDN.

```
dns:
  authserver:
    host: auth
    alternative_fqdns:
      login.4p.movistar.es:
```

2. Create two nested entries, `tls.crt` and `tls.key`, that hold the PEM-formatted public certificate and private key, respectively. Note that the `tls.crt` must include all the certificate chain, including intermediate and root CAs.

```
dns:
  authserver:
    host: auth
    alternative_fqdns:
      login.4p.movistar.es:
        tls.crt: |
          -----BEGIN CERTIFICATE-----
          MIIFjzCCBHegAwIBAgITAPrPqHlByd0zxNtyRIjs8gwruTANBgkqhkiG9w0BAQsF
          ...
          1hGImf/JdiPf4y1xlESbNkA9sNk7eLcMIDZZ6EIgiJsQIrE=
          -----END CERTIFICATE-----
          -----BEGIN CERTIFICATE-----
```

```
    MIIFWzCCA0OgAwIBAgIQTfQrldHumzpMLrM7jRBd1jANBgkqhkiG9w0BAQsFADBm
    ...
    hSjpTUFGSiQrR2JK2Evp+o6AETUkBCO1aw0PpQBPDQ==
    -----END CERTIFICATE-----
    -----BEGIN CERTIFICATE-----
    MIIFVDCCBDygAwIBAgIRAO1dW8lt+99NPs1qSY3Rs8cwDQYJKoZIhvcNAQELBQAw
    ...
    oZ2lS38fL18Aon458fbc0BPHtenfhKj5
    -----END CERTIFICATE-----
  tls.key: |
    -----BEGIN PRIVATE KEY-----
    MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQC5zZxliOiog46F
    ...
    nX1vbuNIqnbUm0F5DepNPZee
    -----END PRIVATE KEY-----
```

Once the tenant is created with this configuration, the service will be accessible in both the default and the alternative FQDNs.

Certificate rotation

When the alternative certificate is close to expire, the OB must manage the certificate renewal and then provision the new certificate in the Kernel Platform, following these steps:

1. Locate the secret where the certificate is stored. It is located in the `baikal-system` namespace and is named `tenant-<TENANT>-<FQDN>-cert-tls`. E.g. `tenant-es-pro-login.4p.movistar.es-cert-tls`.
2. Save the current secret in case a rollback is needed.

   ```
   $ kubectl get secret <secret-name> -o yaml -n baikal-system > backup-secret.yml
   ```

3. Edit the secret and replace the values of the tls.crt and tls.key fields with the Base64 encodings of the PEM-formatted certificate and private key, respectively.

   ```
   $ kubectl edit secret <secret-name> -n baikal-system
   ```

   Tip: You can get the Base64 encoding from a PEM-formatted file running this command:

   ```
   $ base64 --wrap 0 tls.crt
   ```

After the secret has been updated, the new certificate will be served in a few seconds.

Storage accounts lifecycle management and soft delete configuration

The Kernel Platform uses Azure Storage Accounts to store data such as logs, datasets, backups, etc. Some of those storages contain huge amounts of data that need to be sanitized in order to keep size and cost under control. On the other hand, some data is critical and must be mechanisms to recover them from deletion.

Azure provides the Lifecycle management policies feature that allows defining rules to move data from hot tier to cold or archive tier, or delete data based on the last modification time. It also provides the Soft delete feature that allows defining a retention time for deleted data, so they can be recovered in case of accidental deletion.

The Kernel Platform creates a predefined set of rules on some Storage Accounts that can be configured using the config files. There are rules that apply globally to the overall platform and rules that are tenant aware.

The State config file under `config/state` in the State installers allows configuring lifecycle management rules and soft delete for the `datasets` storage account. These rules apply to the whole storage account. The `datasets` storage account comes with a global rule named **sanitize** that moves data to the cool tier after an amount of days after its last modification, which can be configured in the config file. This storage account also have activated the *soft delete* feature, whose retention can be configured in the config file.

```
storage_accounts:
  datasets:
    soft_delete_days_retained: 30
    lifecycle_management_rules:
      sanitize:
        move_to_cool_older_than_days: 180
```

Stop

It is possible to stop a deployed platform. Since it stops all core services, the Kubernetes cluster-autoscaler will be able to reduce the cluster size.

Stopping a platform is particularly useful in two scenarios:

1. To save money when an environment is not actively used (e.g. a development or integration environment).
2. Before taking a snapshot of an existing platform.

Core services

To stop the **core services**:

```
$ ./baikops core stop --infra $INFRA --core $CORE
```

Start

This option starts core services that were previously stopped.

Core services

To start the **core services**:

```
$ ./baikops core start --infra $INFRA --core $CORE
```

⚠️ In case of a **start/stop** procedure failure, you must not delete any namespace manually. Please contact the Kernel Platform Support Team before retrying.

Parallel upgrade

Summary

This document is intended to show the Kernel Platform parallel upgrade process explaining commands step by step to upgrade an existing cluster to a new major/minor version.

Motivation

Kernel Platform is constantly evolving adding new features to be implemented in existing clusters through upgrade procedure. The final result must be an upgraded platform exactly with the same state both at the source and at the target.

This upgrade applies to the following OBs: BR, DE, ES, UK and HISPAM.

Introduction to parallel upgrade procedure

The parallel upgrade can be grouped into the following phases:

- Generate environment: These are tasks related to the initialization of variables to execute the process.
- Upgrade state: Upgrade state cluster in case it is necessary.
- Upgrade pre-window: These are tasks that can be performed days before the nightly window since they do not imply any risk to the service. This phase consist in the deployment of the new platform in parallel with the old one, maintaining the service provided by the old platform.
- Upgrade on-window: These are tasks that must be executed in the nightly window. A short loss of service (up to 10 minutes) could occur during this window. This phase mainly consists in the switch to the new platform.
- Upgrade post-window: These are tasks that can be run once the new cluster is up and running, outside the maintenance window. Its performs cleaning task to reduce costs, like the deletion of the old cluster. Also some migration tasks could be executed in this stage, which ensure all the resources in the platform and business logic are ready to support it.
- Rollback on-window: In case of any fail during the upgrade procedure execution, the Rollback stage is available to back off the new platform and restore the source cluster at the state just before the maintenance window.

There is a glossary of terms which are used during migration with whom we must become familiar:

- **CONFIG**: Refers to the config name whose deployment will be carried out, generally corresponds to `<ob>-<env>`.
- **INFRA_FROM/TO**: Refers to the cluster name from/to the deployment will be carried out, generally corresponds to `<ob>-<env>-<major_version>-<minor_version>`.
- **VERSION_FROM/TO**: Refers to the Kernel Platform version from/to the deployment will be carried out.
- **STATE**: Refers to the name of the set of cloud services where the Kernel Platform stores data to be persisted. It should be the same name as `CONFIG`.
- **VERSION_STATE**: Refers to the Kernel Platform State version used to the State upgrade.
- **CORE**: Refers to the name of the Kernel Platform core set of services to be deployed. It should be the same name as `CONFIG`.
- **AZURE_TENANT_ID**: Identifier of the Azure tenant where will be located all the resources of the Kernel Platform.
- **AZURE_SUBSCRIPTION_ID**: Identifier of the Azure subscription where will be located all the resources of the Kernel Platform.
- **AZURE_CLIENT_ID**: Id of the Service Principal created in Azure Active Directory with full permissions to deploy the Kernel Platform.
- **AZURE_CLIENT_SECRET**: Secret for the Service Principal created in Azure Active Directory.

Detailing phases

Generate environment

In the following steps, the user has to define the necessary variables. The variables cannot be empty. Then, variables will be initialize and finally the docker images used to run the commands will be built.

```
$ ~/parallel-upgrade$ tree -L 1
.
├── fourth-platform-<ob>-<version-from>
├── fourth-platform-<ob>-<version-to>
└── fourth-platform-state-<ob>-<version-state> --> Only when required
```

⚠️ Please, run the following commands in the folder where the installers are located.

Step 1 - Export environment variables

```
$ export AZURE_TENANT_ID='<azure_tenant_id>'
$ export AZURE_SUBSCRIPTION_ID='<azure_subscription_id>'
$ export AZURE_CLIENT_ID='<azure_client_id>'
$ export AZURE_CLIENT_SECRET='<azure_client_secret>'

$ export VAULT_PASSWORD='<installer_vault_password>'
```

Step 2 - Define necessary variables

In the root folder of the installers, execute the following command to define and create the environment file with all the variables used during the upgrade procedure:

```
$ ./fourth-platform-<ob>-<version-to>/utils/create-upgrade-env.sh
```

This script will create the environment file called `upgrade_env` in the root folder of the installers. Please, apply the environment file with the following command:

```
$ source upgrade_env
```

⚠️ The steps detailed in this section have to be executed in every terminal we are going to execute some commands of the installer during upgrade procedure. If `upgrade_env` file is already created, `source` command is enough.

Upgrade state

Step 3 - Upgrade state only if its necessary

⚠️ This upgrade must be done only when there is a version change in the compatibility matrix.

⚠️ All commands specifically referring to upgrade are executed with an alias which is vital since dictates in which cluster each of the steps will be applied. The procedure described below already takes this into account and requires a double check by the user in the most critical steps.

⚠️ The specified commands in this document have to be executed within a `/bin/bash` shell.

⚠️ It is not possible to rotate passwords between upgrades.

⚠️ This flow has been updated and 'baikops upgrade cluster' step has been moved to last step in the flow so state components are upgraded to a version compatible with new kubernetes cluster version previously to upgrading the AKS cluster itself.

```
# Apply config evolutions if needed
$ installer_state && ./baikal-state config prepare-upgrade \
    --state ${STATE} \
    --values config/state-values.yml
```

⚠️ It is possible that no values.yml file has been created because no values are needed to upgrade the configuration. If so, do not execute the next command with the `--values` argument.

```
# Upgrade configuration
$ installer_state && ./baikal-state config upgrade \
    --state ${STATE} \
    --values config/state-values.yml --in-place
```

```
# Upgrade in-place, K8s cluster will not be upgraded in this step
$ installer_state && ./baikal-state create \
    --state ${STATE}
```

```
# K8s upgrade
$ installer_state && ./baikops-state upgrade cluster \
    --state ${STATE}
```

No downtime tasks

As explained before, the tasks related to that section do not imply loss of service and therefore they can be performed days before. Here we go with a brief explanation of what is going to be done:

- Backup DNS record sets.
- Create database backup replica.
- Create the infrastructure.
- Create the core services.
- Configure all tenants.
- Reprocess topics.
- Flush redis cache.

⚠️ All commands specifically referring to upgrade are executed with an alias which is vital since dictates in which cluster each of the steps will be applied. The procedure described below already takes this into account and requires a double check by the user in the most critical steps such as create/delete infrastructure and core.

⚠️ The specified commands in this document have to be executed within a `/bin/bash` shell.

⚠️ It is not possible to rotate passwords between upgrades. The `vault.yml` from `INFRA_FROM` and the services' password from `INFRA_TO` must match, except for the new services' password that has to be included.

ℹ️ All commands in this section are idempotent and they can be executed as many times as wanted (in case of error, for example).

Step 4 - Backup DNS record sets

```
$ installer_to && ./baikops dns backup \
    --infra ${INFRA_FROM} \
    --core ${CORE} \
    --backup-name ${CORE}-${INFRA_FROM}-dns-backup
```

Step 5 - Create database read replicas

```
$ installer_to && ./baikops database replicate \
    --state ${STATE}
```

Step 6 - Volume backup

```
$ installer_to && ./baikops volume backup \
    --state ${STATE} \
    --backup-name ${STATE}-volume-backup
```

Step 7 - Create the infrastructure

```
# Apply evolutions if needed
$ installer_to && ./baikal infra config prepare-upgrade \
    --infra ${INFRA_FROM} \
    --values config/infra-values.yml
```

⚠️ It is possible that no values.yml file has been created because no values are needed to upgrade the configuration. If so, do not execute the next command with the `--values` argument.

```
# Upgrade configuration
$ installer_to && ./baikal infra config upgrade \
    --from-infra ${INFRA_FROM} \
    --to-infra ${INFRA_TO} \
    --values config/infra-values.yml
```

```
$ installer_to && ./baikal infra create \
    --infra ${INFRA_TO} \
    --state ${STATE}
```

Step 8 - Promote database replica

```
$ installer_to && ./baikops database promote-replica \
    --state ${STATE}
```

Step 9 - Upgrade core configuration before topic-migrator create

```
# Apply evolutions if needed
$ installer_to && ./baikal core config prepare-upgrade \
    --infra ${INFRA_FROM} \
    --core ${CORE} \
    --values config/core-values.yml
```

⚠️ It is possible that no values.yml file has been created because no values are needed to upgrade the configuration. If so, do not execute the next command with the `--values` argument.

```
# Upgrade configuration
$ installer_to && ./baikal core config upgrade \
    --from-infra ${INFRA_FROM} \
    --to-infra ${INFRA_TO} \
    --from-core ${CORE} \
    --to-core ${CORE} \
    --values config/core-values.yml
```

```
# Apply evolutions if needed
$ installer_to && ./baikal tenant config prepare-upgrade \
```

```
    --infra ${INFRA_FROM} \
    --tenant ${CORE} \
    --values config/default-tenant-values.yml
```

⚠ It is possible that no values.yml file has been created because no values are needed to upgrade the configuration. If so, do not execute the next command with the `--values` argument.

```
# Upgrade configuration
$ installer_to && ./baikal tenant config upgrade \
    --from-infra ${INFRA_FROM} \
    --to-infra ${INFRA_TO} \
    --from-tenant ${CORE} \
    --to-tenant ${CORE} \
    --values config/default-tenant-values.yml
```

Step 10 - Create topic-migrator

```
$ installer_to && ./baikops kafka topic-migrator create \
    --infra ${INFRA_TO} \
    --old-infra ${INFRA_FROM} \
    --core ${CORE} \
    --state ${STATE}
```

```
# Wait for topic-migrator to complete ongoing migrations
$ installer_to && ./baikops kafka topic-migrator status --wait \
    --infra ${INFRA_TO} \
    --core ${CORE}
```

Step 11 - Create the core services

```
$ installer_to && ./baikal core create \
    --infra ${INFRA_TO} \
    --state ${STATE} \
    --core ${CORE}
```

Step 12 - Configure all tenants

Additional tenants must be published again with each upgrade, similar to the core. All the original data has been migrated with the Kernel Platform's source of truth and redeployment is only necessary to make all those resources, like DNS, available.

⚠ This step must be repeated for each of the tenants that was configured in the previous platform, substituting TENANT name. Example: TENANT1=es-pre-o2, TENANT2=...

```
# Apply evolutions if needed
$ installer_to && ./baikal tenant config prepare-upgrade \
    --infra ${INFRA_FROM} \
    --tenant <TENANT> \
    --values config/tenant-values.yml
```

⚠ It is possible that no values.yml file has been created because no values are needed to upgrade the configuration. If so, do not execute the next command with the `--values` argument.

```
# Upgrade configuration
$ installer_to && ./baikal tenant config upgrade \
    --from-infra ${INFRA_FROM} \
    --to-infra ${INFRA_TO} \
    --from-tenant <TENANT> \
    --to-tenant <TENANT> \
```

```
    --values config/tenant-values.yml
```

```
$ installer_to && ./baikal tenant create \
    --infra ${INFRA_TO} \
    --state ${STATE} \
    --core ${CORE} \
    --tenant <TENANT_1>,<TENANT_2>,...,<TENANT_N>
```

Step 13 - Change the master of consents

⚠ Before the step "Change the master of consents", it is necessary to deploy the corresponding adapters for each tenant (default and secondary ones) only when the OB is the master of consents. This step must be repeated for each of the tenants.

```
$ installer_to && ./baikops job change-master-consents \
    --infra ${INFRA_TO} \
    --core ${CORE} \
    --tenant <TENANT> \
    --consent-api-url <CONSENT_API_URL>
```

Step 14 - Synchronize the legal-entities belonging to the OB

⚠ This step only must be executed when the OB is the master of the consents. This step must be repeated by each of the tenants (default and secondary ones) with their corresponding url.

```
$ installer_to && ./baikops job synchronize \
    --infra ${INFRA_TO} \
    --core ${CORE} \
    --tenant <TENANT> \
    --resource legal-entities \
    --url <LEGAL_ENTITIES_URL_TO_SYNC>
```

Step 15 - Synchronize the pi-scopes belonging to the OB

⚠ This step only must be executed when the OB is the master of the consents. This step must be repeated by each of the tenants (default and secondary ones) with their corresponding url.

```
$ installer_to && ./baikops job synchronize \
    --infra ${INFRA_TO} \
    --core ${CORE} \
    --tenant <TENANT> \
    --resource pi-scopes \
    --url <PI_SCOPES_URL_TO_SYNC>
```

Step 16 - Synchronize the purposes belonging to the OB

⚠ This step only must be executed when the OB is the master of the consents. This step must be repeated by each of the tenants (default and secondary ones) with their corresponding url.

```
$ installer_to && ./baikops job synchronize \
    --infra ${INFRA_TO} \
    --core ${CORE} \
    --tenant <TENANT> \
    --resource purposes \
    --url <PI_SCOPES_URL_TO_SYNC>
```

Step 17 - Reset offsets for all topics except apps, algorithms, and consents

```
$ installer_to && ./baikops job reset-offsets \
```

```
    --core ${CORE} \
    --infra ${INFRA_TO} \
    --skip-topic "apps" \
    --skip-topic "algorithms" \
    --skip-topic "consents"
```

⚠️ RUN THE FOLLOWING COMMAND ONLY WHEN A **ROLLBACK HAS BEEN APPLIED PREVIOUSLY** AND THE CORE PLATFORM HAS BEEN REDEPLOYED. This command will reset the consumer groups of baikal.core.consents.personal topic and it implies long processing time:

```
$ installer_to && ./baikops job reset-offsets \
    --core ${CORE} \
    --infra ${INFRA_TO} \
    --only-topic 'consents'
```

ℹ️ In case of errors in the old platform after applying this procedure, it must be considered as a disaster and the disaster recovery procedure must be applied, using the database and volume backups created in this procedure.

⚠️ You must wait for some time checking the baikal.core.apis and baikal.core.datasets topics lag to concrete the end of the reprocessing step. Until all the lag have not been consumed in those topics, you shouldn't go into the next step. OPTIONAL - You can use the following command to check:

```
$ installer_to && ./baikops job reset-offsets-wait \
    --core ${CORE} \
    --infra ${INFRA_TO} \
    --topic "apis" \
    --topic "datasets"
```

Step 18 - Reprocessing all apps and algorithms topics

```
$ installer_to && ./baikops job reset-offsets \
    --core ${CORE} \
    --infra ${INFRA_TO} \
    --only-topic "apps" \
    --only-topic "algorithms"
```

⚠️ In the same way than previous step, you must check the reprocessing status through the lag in baikal.core.apps and baikal.core.algorithms topics. OPTIONAL - You can use the following command to check:

```
$ installer_to && ./baikops job reset-offsets-wait \
    --core ${CORE} \
    --infra ${INFRA_TO} \
    --topic "apps" \
    --topic "algorithms" \
    --topic "executions-schedules" \
    --topic "executions-streams"
```

Step 19 - Delete deprecated algorithm schedules

```
$ installer_to && ./baikops job delete-deprecated-algorithm-schedules \
    --infra ${INFRA_TO} \
    --core ${CORE}
```

Step 20 - Flush redis cache

After all these steps has been executed and the platform is fully up & running, it is recommended to make a deep flush of the cache in the platform to avoid any kind of issue with cached data during upgrade procedure. The following command should be executed:

```
$ installer_to && ./baikops cache flush \
    --infra ${INFRA_TO} \
    --core ${CORE}
```

Step 21 - Check authserver config

Check if there is any misalignment in Authserver config as a result from operations not persisted in Kernel Configuration Service.

```
$ installer_to && ./baikops diff authserver \
    --infra_to ${INFRA_TO} \
    --infra_from ${INFRA_FROM} \
    --core ${CORE}
```

Step 22 - Check api resources

You can check if all API resources has been migrated by executing the following command. You should do this for each tenant:

```
$ installer_to && ./baikops diff api \
    --infra ${INFRA_TO} \
    --tenant <TENANT>
```

ℹ️ You can also check detailed differences between the old and the new platform by adding the `--detailed` flag. Some changes could be expected, but it is recommended to check the differences before applying the downtime tasks.

Step 23 - Check docker image algorithms

You can verify that the algorithm images match between different tenants. If there are differences, you can verify which images differ and in which tenants they were produced:

```
$ installer_to && ./baikops diff algorithm \
    --infra ${INFRA_TO} \
    --core ${CORE}
```

At this time the new platform is up & running, although it is not providing public service in global endpoints.

⚠️ After this phase of the parallel upgrade procedure completes successfully, it will be necessary to deploy the corresponding versions of different components: corresponding OB's Adapters, Whatsapp service (without bots migration), and ATC to have completed the full upgrade procedure. Whatsapp and ATC only for required OBs.

Downtime tasks

The tasks performed in this section must be carried out during the maintenance window since it involves service downtime. Below it can be found a summary about the steps to be executed during this phase:

- Switch services to active mode.
- Switch DNS.

⚠️ All commands specifically referring to upgrade are executed with an alias which is vital since determines in which cluster each of the steps will be applied. All the steps in the procedure described below already takes this into account and requires a double check by the user in the most critical steps such as create/delete infrastructure and core.

⚠️ The specified commands in this document have to be executed within a `/bin/bash` shell.

ℹ️ Bear in mind if this section will be performed from a different shell that the one used in previous steps, you have to execute again section Generate environment.

ℹ️ All commands in this section are idempotent and they can be executed as many times as wanted (in case of error, for example).

Step 24 - Disable algorithm streams in (old core)

```
$ installer_to && ./baikops job algorithm-streams \
    --infra ${INFRA_FROM} \
    --core ${CORE} \
    --tenant ${CORE} \
    disable
```

⚠️ This step must be repeated for each of the tenants that was configured in the previous platform, substituting TENANT name. Example: TENANT1=es-pre-o2, TENANT2=...

```
$ installer_to && ./baikops job algorithm-streams \
    --infra ${INFRA_FROM} \
    --core ${CORE} \
    --tenant <TENANT> \
    disable
```

Step 25 - Configure services as passive in old core

```
$ installer_to && ./baikops service mode-set \
    --infra ${INFRA_FROM} \
    --core ${CORE} \
    --mode passive \
    --log-level ERROR
```

Step 26 - Copy streaming executions from one cluster to another

```
$ installer_to && ./baikops job copy-streaming-executions \
    --infra_from ${INFRA_FROM} \
    --infra_to ${INFRA_TO} \
    --core ${CORE}
```

Step 27 - Configure services as active in new core

```
$ installer_to && ./baikops service mode-set \
    --infra ${INFRA_TO} \
    --core ${CORE} \
    --mode active \
    --log-level INFO
```

Step 28 - Enable algorithm streams in (new core)

```
$ installer_to && ./baikops job algorithm-streams \
    --infra ${INFRA_TO} \
    --core ${CORE} \
    --tenant ${CORE} \
    enable
```

⚠️ This step must be repeated for each of the tenants that was configured in the previous platform, substituting TENANT name. Example: TENANT1=es-pre-o2, TENANT2=...

```
$ installer_to && ./baikops job algorithm-streams \
    --infra ${INFRA_TO} \
```

```
    --core ${CORE} \
    --tenant <TENANT> \
    enable
```

Step 29 - Switch DNS

```
$ installer_to && ./baikops dns switch \
    --core ${CORE} \
    --infra ${INFRA_TO}
```

Step 30 - Status DNS

```
$ installer_to && ./baikops dns status \
    --infra ${INFRA_TO} \
    --tenant ${CORE}
```

At this time the new platform is on service and it is attending all the requests send by the clients.

⚠️ If your platform is also providing Whatsapp service, in this phase of the parallel upgrade procedure the bots should be migrated from the previous platform to the new one, to start providing service.

⚠️ If you have configured alternative FQDN in the tenant config, OB-branded FQDN must be setup in the OB's DNS service with a CNAME record pointing to the default FQDN (e.g. auth.TENANT.baikalplatform.com).

Post downtime tasks

The following steps could be run outside the maintenance window.

Once the new target cluster is up and running as expected, source cluster can be destroyed and the core database must be migrated to flexible server.

- Delete the old core services.
- Delete the old infrastructure.
- Clean inactive databases.

🛑 This will destroy the old cluster, there is no turning back to the old cluster from this point.

⚠️ All commands specifically referring to migration are executed with an alias which is vital since dictates in which cluster each of the steps will be applied. The procedure described below already takes this into account and requires a double check by the user in the most critical steps such as create/delete infrastructure and core.

⚠️ The specified commands in this document have to be executed within a `/bin/bash` shell.

ℹ️ All commands in this section are idempotent and they can be executed as many times as wanted (in case of error, for example).

Step 31 - Delete topic-migrator

```
$ installer_to && ./baikops kafka topic-migrator delete \
    --infra ${INFRA_TO} \
    --core ${CORE} \
    --state ${STATE}
```

```
$ installer_to && ./baikops kafka topic-migrator clean \
    --infra ${INFRA_TO} \
    --old-infra ${INFRA_FROM} \
    --core ${CORE} \
    --state ${STATE}
```

Step 32 - Delete old core services

```
$ installer_from && ./baikal core delete \
    --infra ${INFRA_FROM} \
    --state ${STATE} \
    --core ${CORE}
```

```
$ installer_from && ./baikal core config delete \
    --infra ${INFRA_FROM} \
    --core ${CORE}
```

```
$ installer_from && ./baikal tenant config delete \
    --infra ${INFRA_FROM} \
    --tenant ${CORE}
```

⚠️ This step must be repeated for each of the tenants that was configured in the next platform, substituting TENANT name. Example: TENANT1=es-pre-o2, TENANT2=...

```
$ installer_from && ./baikal tenant config delete \
    --infra ${INFRA_FROM} \
    --tenant <TENANT>
```

Step 33 - Activate jwt cronjobs and Delete old Consumer groups from Kafka

```
$ installer_to && ./baikops job enable-jwt-cron \
    --infra ${INFRA_TO} \
    --core ${CORE} \
    --suspend false
```

```
$ installer_to && ./baikops kafka delete-old-consumers-groups \
    --infra_to ${INFRA_TO} \
    --infra_from ${INFRA_FROM} \
    --core ${CORE}
```

Step 34 - Destroy old infrastructure

- Remove state lock:

⚠️ Remove state lock is only required while installer_from is before 2024.19 release

```
az login --service-principal --username ${AZURE_CLIENT_ID} --password ${AZURE_CLIENT_SECRET} --tenant ${AZURE_T
ENANT_ID}
az lock delete \
    --name "baikal-${STATE}-state-rg" \
    --resource-group "baikal-${STATE}-state-rg" \
    --subscription "${AZURE_SUBSCRIPTION_ID}"
```

- Destroy old infrastructure:

```
$ installer_from && ./baikal infra delete \
    --infra ${INFRA_FROM} \
    --state ${STATE}
```

```
$ installer_from && ./baikal infra config delete \
```

```
    --infra ${INFRA_FROM}
```

- Restore state lock:

⚠️ Restore state lock is only required while installer_from is before 2024.19 release

```
az login --service-principal --username ${AZURE_CLIENT_ID} --password ${AZURE_CLIENT_SECRET} --tenant ${AZURE_T
ENANT_ID}
az lock create \
    --name "baikal-${STATE}-state-rg" \
    --resource-group "baikal-${STATE}-state-rg" \
    --lock-type "CanNotDelete" \
    --notes "Managed by Pynstaller." \
    --subscription "${AZURE_SUBSCRIPTION_ID}"
```

Step 35 - Clean up inactive databases

In case there are any locks in Azure resources, it is necessary to remove them from the Azure portal before running this step.

```
$ installer_to && ./baikops database clean-inactive \
    --state ${STATE}
```

Rollback

🆘 **Rollback can only be performed during the maintenance window**

This section is only executed if something went wrong during the migration. If a back off is needed it must be done during the maintenance window. Here it can be found a brief explanation of what is going to be done:

- Switch DNS record sets.
- Switch services to passive mode.

⚠️ All commands specifically referring to migration are executed with an alias which is vital since dictates in which cluster each of the steps will be applied. The procedure described below already takes this into account and requires a double check by the user in the most critical steps such as create/delete infrastructure and core.

⚠️ The specified commands in this document have to be executed within a `/bin/bash` shell.

ℹ️ All commands in this section are idempotent and they can be executed as many times as wanted (incase of error, for example).

ℹ️ Bear in mind, if this section will be performed from a different shell that the one used in previous steps you have to executed again section Generate environment.

Step 36 - Switch DNS record sets

```
$ installer_to && ./baikops dns switch \
    --core ${CORE} \
    --infra ${INFRA_FROM}
```

Step 37 - Status DNS

```
$ installer_to && ./baikops dns status \
    --infra ${INFRA_FROM} \
    --tenant ${CORE}
```

Step 38 - Disable algorithm streams in (new core)

```
$ installer_to && ./baikops job algorithm-streams \
    --infra ${INFRA_TO} \
    --core ${CORE} \
    --tenant ${CORE} \
    disable
```

⚠️ This step must be repeated for each of the tenants that was configured in the previous platform, substituting TENANT name. Example: TENANT1=es-pre-o2, TENANT2=...

```
$ installer_to && ./baikops job algorithm-streams \
    --infra ${INFRA_TO} \
    --core ${CORE} \
    --tenant <TENANT> \
    disable
```

Step 39 - Switch services to passive in new_core

```
$ installer_to && ./baikops service mode-set \
    --infra ${INFRA_TO} \
    --core ${CORE} \
    --mode passive \
    --log-level ERROR
```

Step 40 - Switch services to active in old_core

```
$ installer_to && ./baikops service mode-set \
    --infra ${INFRA_FROM} \
    --core ${CORE} \
    --mode active \
    --log-level INFO
```

Step 41 - Enable algorithm streams in (old core)

```
$ installer_to && ./baikops job algorithm-streams \
    --infra ${INFRA_FROM} \
    --core ${CORE} \
    --tenant ${CORE} \
    enable
```

⚠️ This step must be repeated for each of the tenants that was configured in the previous platform, substituting TENANT name. Example: TENANT1=es-pre-o2, TENANT2=...

```
$ installer_to && ./baikops job algorithm-streams \
    --infra ${INFRA_FROM} \
    --core ${CORE} \
    --tenant <TENANT> \
    enable
```

At this point the old cluster is on service and it is attending all the requests send by the clients.

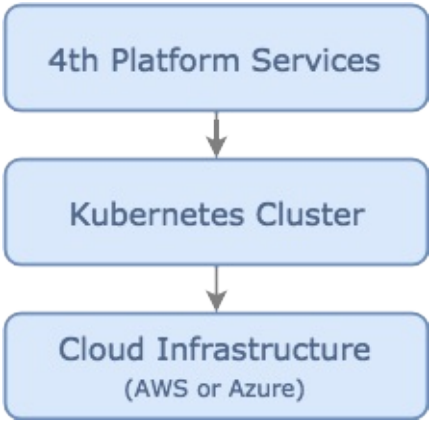
⚠️ Although the old cluster has no changes after the rollback procedure, the operations performed during the upgrade procedure has been persisted in the state. For this reason, we encourage avoiding the deletion of resources (like apps, apis, scopes, etc.) because these operations will be performed again in the next upgrade.

Kernel Platform Operation Guide

This Operation guide describes how the Kernel Platform Service can be operated through the different available commands and tools. It is intended to provide useful information for the Kernel Platform operation teams. However, development teams could also find this guide useful not only to give support to the operation teams, but also to better understand the physical and logical deployment architecture and the internal information flows.

Deployment architecture

The Kernel Platform can be deployed on Azure. We use Kubernetes (provided by AKS, the Azure Kubernetes Service) to orchestrate the containers and other Azure services that composes the Kernel Platform. For these reasons, the Kernel Platform cannot be deployed at other cloud provider.



In this section we provide details about the cloud infrastructure, the Kubernetes cluster and the Kernel Platform services, focusing on how it works, how to manage it and how to fix the most common issues.

Remember that you can get more information or answer any questions you might have on https://qna.baikalplatform.com/, the Kernel Platform Community Support center.

Basic Concepts

- **Baikal** is the internal codename for the Kernel Platform.
- A **config**, also called a deployment config, is a YAML file used during the Kernel Platform installation to configure the infrastructure and services.
- **Kubernetes** is an open-source system for automating deployment, scaling, and management of containerized applications. This guide is not intended to be a Kubernetes tutorial because the official documentation is very complete. We try to detail what makes the Kernel Platform Kubernetes cluster special, and how to debug and solve common issues.

Cloud Infrastructure

Computing

The Kernel Platform services are deployed on virtual machines[1] in Azure. Different types of nodes are used to schedule different kinds of loads (e.g. cpu-intensive, memory-intensive or IO-intensive tasks). Each group is called an **agent pool**, node pool or instance group, and all nodes in a group share the same configuration.

Currently, there are the following agent pools:

- **system**: For system tasks, mostly managed by AKS (e.g. coredns).
- **common**: For generic tasks, often stateless, that don't have specific hardware requirements.
- **kafka**: For Kafka clusters that has some specific requirements (e.g. IO) and we want to keep it separated from the rest of the processes (they are only present at the state cluster)

If there are multiple available zones configured for the Kernel Platform, the common and kafka nodepools creates one agent pool per availability zone, appending the number of the availability zone to the agent pool name (e.g. common1 and common2).

Remember that Kubernetes **master** nodes contain the Kubernetes control plane and are managed by Azure, so no access is provided to these nodes.

Moreover, the processing environment runs algorithms in a dedicated set of agent pools that can be configured in the config (`infrastructure_processing.compute`) and the algorithm resource usage inside `algorithm_tiers` . For example:

```
infrastructure_processing:
  compute:
    drivers:
      type: "Standard_F8s_v2"
    genm:
      type: "Standard_DS3_v2"
    genmlow:
      type: "Standard_DS3_v2"

  algorithm_tiers:
    general-medium:
      pool_name: "genm"
      algorithm_memory_gb: 30
      algorithm_cpu: "7000"
    general-medium-lowpriority:
      pool_name: "genmlow"
      algorithm_memory_gb: 30
      algorithm_cpu: "7000"
```

Each algorithm's author must choose the agent pool they want to use to run their algorithms.

⚠️ We try to make nodes as immutable as possible. Accessing the cluster nodes is strongly discouraged and installing any kind of process or agent is not supported and can cause issues in the Kernel Platform.

We will come back to the agent pools in the Kubernetes section.

> [1]. We use the terms virtual machine, machine, instance and node interchangeably. ↩

Networking

Network architecture

The Azure platform automatically creates and configures the virtual network resources. All nodes are created in a private VNET so access from the Internet is forbidden.

The cluster uses Azure Container Networking Interface (CNI) networking as network model.

With Azure Container Networking Interface (CNI), every pod gets an IP address from the subnet and can be accessed directly. These IP addresses must be unique across your network space, and must be planned in advance. Each node has a configuration parameter for the maximum number of pods that it supports. The equivalent number of IP addresses per node are then reserved up front for that node.

Current Azure deployments use a single VNET and up to 5 subnets:

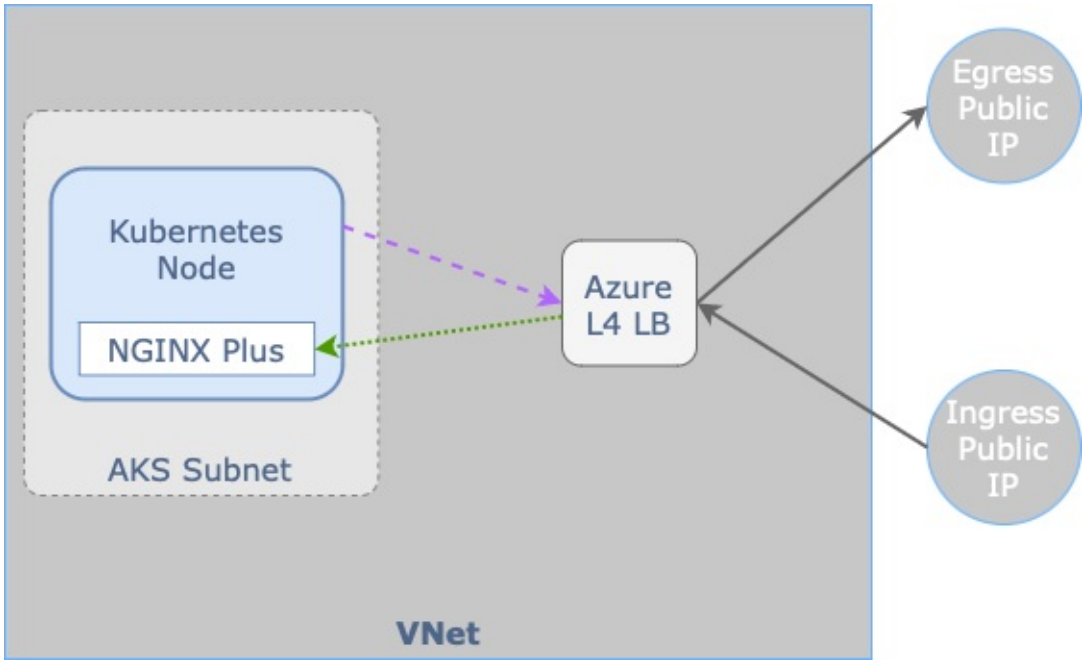
- 2 /12 subnets for the services Kubernetes clusters. There are 2 because could be 2 clusters during parallel upgrades.
- 1 /16 subnet for the state Kubernetes cluster.

The high availability of the VNET is provided by Azure, so is not managed by the Kernel Platform and is opaque to Kernel Platform operators.

Also, each node has 80 pods per node as limit. See https://docs.microsoft.com/en-us/azure/aks/configure-azure-cni for further information.

Ingress HTTP traffic is routed to the cluster via Azure Load Balancer managed by AKS, that has an ingress public IP assigned. Egress traffic is sent using ALB too, and uses a different public IP.

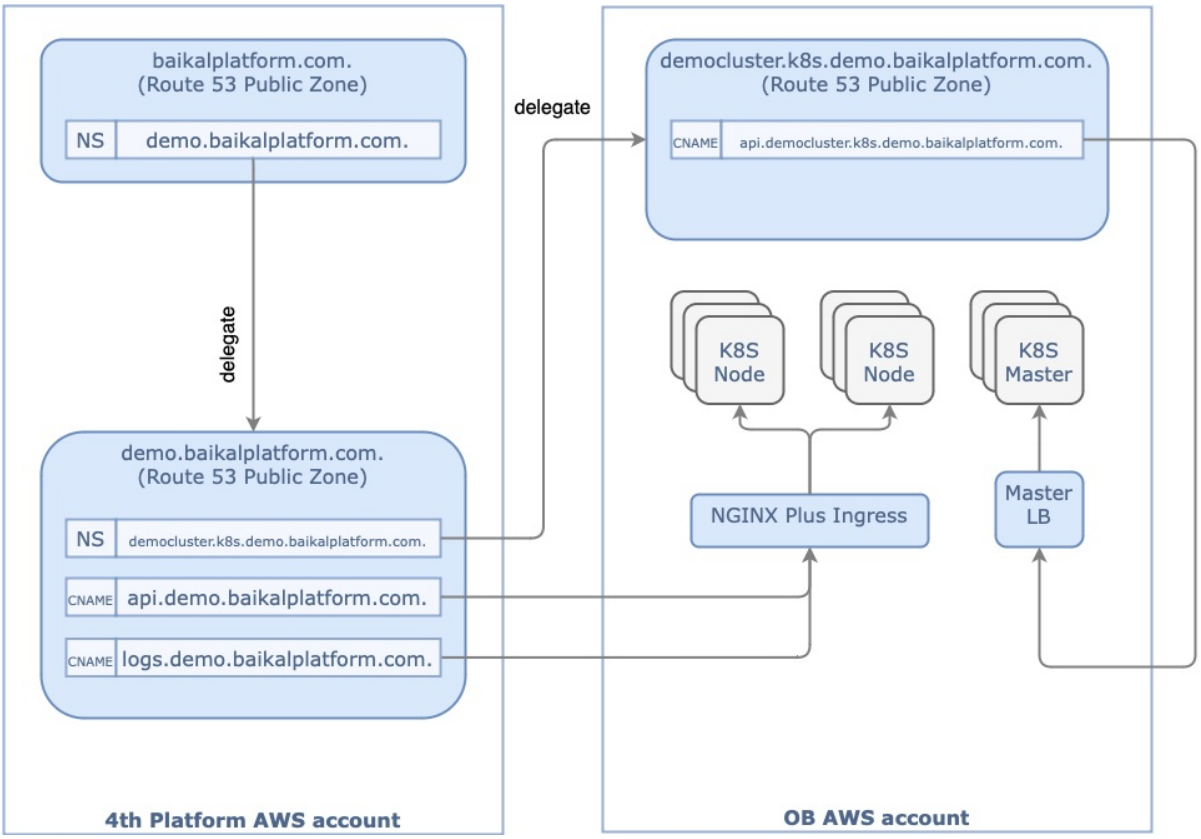
Ingress traffic is TLS encrypted (uses HTTPS). The TLS termination is done by NGINX Plus Ingress, that sends unencrypted traffic to the backend services. Note that the internal traffic between services is not encrypted.



DNS architecture

The top domain name is `baikalplatform.com` and it is managed in a global Route53 service account managed by the Kernel Platform Team. All deployed platforms hang from it as nested subdomains.

There is only one DNS zone, delegated from the Kernel Platform Global AWS account, for the Kernel Platform services.



Cluster named demo and westus region assumed in above diagram.

Ingress: access ranges

Access to different services in the platform can be restricted not only to authenticated users (basic authentication is always enabled) but also to specific origin IP ranges. For example, you might want to restrict access to some monitoring services (e.g. Grafana and Kibana) to users in your network only. By default, access to the Kernel Platform services is not restricted by origin (the CIDR " `0.0.0.0/0` " means any IP address). This setup also presents some benefits to simplify monitoring, operations and support from remote teams, including devs in the Kernel Platform Team that use monitoring information to improve the Kernel Platform.

```
service_whitelist_source_ranges:
  alertmanager:
    - 0.0.0.0/0
  prometheus:
    - 0.0.0.0/0
  apigw:
    - 0.0.0.0/0
  authserver:
    - 0.0.0.0/0
  grafana:
    - 0.0.0.0/0
  kibana:
    - 0.0.0.0/0
  portal:
    - 0.0.0.0/0
  elasticsearch:
    - 0.0.0.0/0
```

The " `service_whitelist_source_ranges` " can be changed in the config and redeployed anytime without service disruption. It is reflected at:

- Ingress " `ingress.kubernetes.io/whitelist-source-range` " annotation
- Service " `.spec.loadBalancerSourceRanges` " property

For example, to check the range of IPs allowed to access Grafana:

```
$ kubectl describe ingress grafana
...
Annotations:
  ingress.kubernetes.io/whitelist-source-range:      0.0.0.0/0
```

ℹ️ The IPs used to expose the Kernel Platform APIs have specific considerations regarding their lifecycle. Ingress IPs are static and their lifecycle is bound to the infrastructure.

⚠️ **These IPs ARE NOT customizable by the Kernel Platform operators**, they are sourced randomly from an Azure IP address pool.

⚠️ **These IPs ARE NOT transferable between infrastructures**. This means that a parallel upgrade will always create a new IP, because previous IP can't be detached without compromising the rollback procedure. The Kernel Platform provides 2 Ingress IPs and 2 Egress IP ranges to switch between both of them.

Check the following post in the Kernel Platform Community Support center for more information on this topic.

Egress

The Kernel Platform needs to access external services on the Internet. Not only to access the OB APIs, but also to download the Docker images from the different Docker registries (public registries and the private Kernel Platform registries).

The Load Balancers have multiple public IPs and all egressing traffic goes through them. Review Scenario 2 in https://docs.microsoft.com/en-us/azure/load-balancer/load-balancer-outbound-connections for wider information.

The Kernel Platform Team considers that relying on a static set of IP addresses is as a bad practice for cloud services because it is not safer than a 2-way-SSL communication channel to guarantee the identity of both sides, and makes maintenance much harder.

DNS

The top domain name is " `baikalplatform.com` " by default, and it is managed by the Kernel Platform Team. From it, separate DNS zones are created for each " `$TENANT` " and delegated resolution from top domain. Within these zones, the records above are stored along with additional meta records used for deployment coordination.

Core Services

The Kernel Platform automatically creates the following subdomains under " `$TENANT.baikalplatform.com` " and registers all required DNS entries in Route53, pointing to Kernel public external IP, hosted by Azure Load Balancer. Then ALB forwards the requests to NGINX Ingress Plus that is the entry point to k8s cluster.

- `www`
- `api`
- `auth`
- `dashboard`
- `metrics`
- `alerts`
- `logs`
- `rawlogs`
- `jdbc`

Service records' type is "A" to point to Azure LB IPv4. There is not support for IPv6 in the Kernel Platform yet.

Infrastructure Control Plane (Kubernetes)

There is another DNS domain pointing to an additional Load Balancer that exposes Kubernetes API on the master nodes. It is stored in the kubeconfig file and is a record under CloudApplication Azure regional managed domain, which follows the pattern `$PLATFORM.$region.cloudapp.azure.com`.

Storage

The Kernel Platform uses three types of storage: **local storage, persistent storage (external volumes) and object storage.**

All nodes have a small **local disk** that is used for the operating system and ephemeral storage.

For **persistent storage**, used by some stateful services, an external SSD volume is dynamically attached to the node when needed. This way, when a node fails, its data can be reattached to another node in a short period of time. In any case, to provide a second layer of protection against data loss, all the services that store data in the Kernel Platform (Kafka, Zookeeper) are designed to form a cluster in a way that, when a volume is lost, the data is automatically replicated from another node in the cluster.

The Kernel Platform uses Premium SSD Managed Disks, backed by SSDs, with locally redundant storage, keeping three copies of the data within a single region. Azure manages the storage accounts that you use for your VM disks.

All external volumes are encrypted at rest and configured to be retained when they are removed from Kubernetes (see how to manage persistent volumes in the Kubernetes section). Retaining them guarantees an additional point of recovery that can be used as the last resort in case something goes wrong.

ℹ️ Check the following posts in the Kernel Platform Community Support center for more information on this topic:

- How to bind a existing cloud volume to a new Persistent Volume (PV)
- How to bind a existing Persistent Volume (PV) to a new Persistent Volume Claim (PVC)

The size of the external volumes is configured in the deployment config. Increasing the size of the volumes is not a supported operation by the Kernel Platform installer, but can exceptionally be done manually. Support for disk resize will improve in future versions of Kubernetes. Please follow the official Kubernetes guide for resizing Persistent Volume Claims (PVCs). Remember that changes done to the infrastructure that are not reflected in the deployment config can be lost **but these changes are incompatible with an in-place deployment** due to inmutable nature of StatefulSet properties.

The operation-manager is an exception, and uses a small amount of the local storage (a few GBs) as part of its multi-level cache of operations. This service tracks the process of all the operations in the Kernel Platform and does not store personal information.

The Kernel Platform also uses the **object storage** (Azure Blob Storage) to store some objects:

- Terraform state. Information in this bucket contains the infrastructure status managed by Terraform. It should never be manually removed.
- Datasets. Information about all datasets. It currently stores not only the ingested datasets, but also the platform KPIs that are not modelled as a dataset yet.
- Backups. Zookeeper and Kafka backups (if enabled) are stored here.
- Algorithms. An artifact repository containing the implementation of the algorithms that run in the Kernel Platform processing environment.
- Aura conversations. It is an ad-hoc solution created for Aura. It will also be removed to use the official Kernel Platform data ingestion mechanism.

All data stored in the **object storage** is encrypted at rest with a key managed by Azure. The traffic between the object storage and the Kubernetes cluster is also encrypted (HTTPS).

This is an example an storage account encryption configuration:

```
az storage account show --ids <storage-account-id>
{
```

```
  ...
  "encryption": {
    "encryptionIdentity": null,
    "keySource": "Microsoft.Storage",
    "keyVaultProperties": null,
    "requireInfrastructureEncryption": null,
    "services": {
      "blob": {
        "enabled": true,
        "keyType": "Account",
        "lastEnabledTime": "xxx"
      },
      "file": {
        "enabled": true,
        "keyType": "Account",
        "lastEnabledTime": "xxx"
      },
      "queue": null,
      "table": null
    }
  },
  ...
}
```

High availability

Using several **availability zones** to the Kernel Platform is a basic mechanism to increase the availability of the platform when failures occur. Three availability zones is the minimum required to guarantee high availability for the Kernel Platform services that need quorum to work (zookeeper).

There is a `ha_zones` attribute in the `infrastructure` section of the deployment config:

```
infrastructure:
  ha_zones: 3
```

You can choose the number of availability zones you want to use. Three HA zones is recommended for production environments to guarantee that quorum-based system such as zookeeper can run even if an availability zone suffers an outage. Also, you can set `ha_zones` to 0 to disable the use of availability zones.

The Kernel Platform is designed for resiliency and integrates many self-healing capabilities. Nodes can be restarted anytime or destroyed if they are misbehaving. However, it is recommended to drain the nodes if it is possible before restarting them.

Uninstalling the infrastructure

It is possible to remove an environment when you no longer use it, and redeploy it only when needed. This operation is not reversible and removes almost ALL the cloud resources and some of the data stored in the platform. The process is described in the "Undeployment" section of the Deployment guide.

Kubernetes cluster

The Kubernetes clusters are created on the Azure cloud using Azure Kubernetes Service (AKS).

The Kernel Platform uses two Kubernetes clusters, one for the services and other for the platform state.

Connecting to Kubernetes

The **kubectl** command-line tool is the basic tool to operate the Kubernetes clusters. It uses **kubeconfig files**, that contain all the required information (endpoints, certificates, etc) to connect with the Kubernetes API and manage the cluster in a secure way. A default (root) kubeconfig file with full access to the cluster is created the first time you install the Kernel Platform. It is stored and managed by Azure and can be obtained with Azure CLI.

This is the command to obtain the services cluster kubeconfig:

```
$ az aks get-credentials --name $INFRA --resource-group baikal-$INFRA-cluster-rg --subscription $AZURE_SUBSCRIPTION_ID
```

And this is the one to obtain the state cluster kubeconfig:

```
$ az aks get-credentials --name $STATE --resource-group baikal-$STATE-state-rg --subscription $AZURE_SUBSCRIPTION_ID
```

Do NOT use the default kubeconfig file to manage the cluster. Use it to create new users with limited permissions instead.

Once your personal kubeconfig file is ready, you need to let `kubectl` know its location. By default, kubectl looks for a file named `config` in the `$HOME/.kube` directory. However, you can specify other kubeconfig files by setting the `KUBECONFIG` environment variable or passing the flag `--kubeconfig` to kubectl.

```
$ export KUBECONFIG=/path/to/kubeconfig.yml
```

⚠️ For security reasons, kubeconfig files are personal and must not be shared. Each action a user executes on the cluster is logged. If your kubeconfig file is compromised, you must report it, delete the user and create a new one.

ℹ️ You can get more information about kubeconfig files at the official Kubernetes documentation

Kubernetes automation with Azure Cloud

Kubernetes automates certain tasks such as creating a Load Balancer for a service or mounting a disk on a node as a persistent volume for a pod. Authentication against Azure APIs is done using some Managed Identities which are created during Kernel Platform deployment.

Also, a service principal is created during the Kernel Platform deployment. It is used by some internal services to access to the Azure API.

⚠️ Please don't change the password, delete the Service Principal or remove its Contributor role because credentials won't be automatically updated in the AKS cluster. The "Security" section describes the procedure to change the Service Principal credentials in Azure.

Kubernetes Namespaces

Kubernates Namespaces are a way to divide and organize cluster resources.

You can list the existing namespaces running `kubectl get namespaces`. In the Kernel Platform services cluster you will find the following ones:

- Namespaces used by the Kernel Platform services:
  - **baikal-$CORE**: contains the Kernel Platform core services.
  - **baikal-system**: contains the Kernel Platform system services (prometheus, alertmanager, node-exporter, fluent-bit, fluentd, elasticsearch, kube-static-metrics, kibana). These services are described in this guide.
  - **baikal-infra**: contains customized services associated to the Azure Kubernetes Service infrastructure.
  - Each algorithm in the Kernel Platform processing environment runs in separate namespaces.
- Namespaces used by Kubernetes:
  - **kube-system**: for objects created by the Kubernetes system. The Kernel Platform also deploy some objects into this namespace that are very tied to the infrastructure.
  - **kube-public**: readable by all users, should be empty.
  - **kube-node-lease**
- Default namespace (**default**) for objects with no other namespace.

In the Kernel Platform state cluster you will find the following namespaces:

- Namespaces used by the Kernel Platform state:
  - **baikal-state**: contains the Kernel Platform state services.
  - **baikal-system**: contains the Kernel Platform system services.
  - **velero**: contains the backup service used in upgrades and in disaster recovery procedures.
- Namespaces used by Kubernetes: the same that in the services cluster.

In most situations you will need to use **baikal-$CORE**. You can use the `--namespace` flag in `kubectl` to specify which namespace you are referring to. For example, to get the pods in the Kernel Platform core:

```
$ kubectl get pods --namespace baikal-$CORE
```

Remember that some low-level resources, such as nodes and persistent volumes, are not in a namespace.

Kubernetes networking

We use **Azure CNI**, as a networking solution for Kubernetes. You can find more information about networking in a Kubernetes cluster in the official documentation.

DNS service discovery

There is a `coredns` service in the namespace `kube-system` in charge of DNS resolutions in the cluster. The associated deployment is able to automatically autoscale based on the cluster size. This service is managed by AKS.

You can get the `coredns` running pods filtering by one of their labels:

```
$ kubectl get pods -l k8s-app=kube-dns --namespace kube-system

NAME                       READY   STATUS    RESTARTS   AGE
coredns-544d979687-2g2qv   1/1     Running   0          10h
coredns-544d979687-6wwf4   1/1     Running   0          8h
coredns-544d979687-872fv   1/1     Running   0          10h
```

If `coredns` pods aren't responding for some reason, you may try to perform a rolling restart without downtime using:

```
$ kubectl rollout restart --namespace kube-system deployment coredns
```

ℹ️ Please follow the official Kubernetes guide for troubleshooting DNS service discovery healthiness.

Network Policies

A network policy is a set of network traffic rules applied to a group of pods in the Kubernetes cluster.

This is the official supported method to protect communications between services in the Kubernetes cluster, instead of Azure Network Security groups (NSG). Currently it is implemented for the access flow, including NGINX Plus Ingress, APIGw and AuthServer elements. It is implemented using Calico as Network Policy manager according to Secure traffic between pods using network policies in Azure Kubernetes Service.

Deployed network policies can be listed using the following commands on both system and core namespaces:

```
$ kubectl get netpol -n baikal-$CORE
NAME                    POD-SELECTOR
apigw                   app=apigw
authserver              app=authserver
prometheus-apigw        app=apigw
prometheus-authserver   app=authserver

$  kubectl get netpol -n baikal-system
NAME                   POD-SELECTOR
internet-nginx-ingress          app=nginx-ingress
prometheus-nginx-ingress     app=nginx-ingress
nginx-ingress-alertmanager   app=nginx-ingress
nginx-ingress-apigw          app=nginx-ingress
nginx-ingress-apimock        app=nginx-ingress
nginx-ingress-authserverui   app=nginx-ingress
nginx-ingress-elastic        app=nginx-ingress
nginx-ingress-grafana        app=nginx-ingress
nginx-ingress-inboundgw      app=nginx-ingress
nginx-ingress-kibana         app=nginx-ingress
nginx-ingress-kubeapi        app=nginx-ingress
nginx-ingress-portalbe       app=nginx-ingress
nginx-ingress-portalui       app=nginx-ingress
nginx-ingress-spark          app=nginx-ingress
nginx-ingress-thanos         app=nginx-ingress
```

A detailed definition on the policy can be obtained, eg. for nginx-ingress-apigw flow:

```
$  kubectl describe netpol nginx-ingress-apigw -n baikal-system
Name:         nginx-ingress-apigw
Namespace:    baikal-system
Created on:   2022-04-22 08:36:51 +0200 CEST
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     app=nginx-ingress
  Not affecting ingress traffic
  Allowing egress traffic:
    To Port: 8080/TCP
    To Port: 8000/TCP
    To:
      NamespaceSelector: stack=core
      PodSelector: app=apigw
  Policy Types: Egress
```

Extending network policies coverage to other platform flows is not in the scope of this release and will be considered as part of the platform roadmap.

Activation an deactivation of this feature is controlled by enable_network_policies configuration parameter.

In-transit traffic encryption

In addition to network policies, enabling Calico can also be used to provide in-transit encryption for traffic between pods in different AKS nodes. To enable this feature, it is required to activate the `enable_encryption_in_transit` configuration parameter.

Activation of this feature is independent from network policies. It can be activated even when network policies feature is disabled.

Extra information about in-transit encryption support in Calico is available here.

Kubernetes objects

Working with pods

You can list all pods in a given namespace along with additional metadata (the node where the pod is allocated, its age, etc):

```
$ kubectl get pods -n baikal-$CORE -o wide
```

Pods can be in different statuses:

- `Running`
- `Completed` : some pods (e.g. jobs) have a reduced lifespan. They transition to the `Completed` status when they finish. Kubernetes eventually removes them form the list of pods.
- `Pending` : Kubernetes is still trying to find a valid place to run the pod.
- Others

ℹ️ You can get more information about working with pods in the Kubernetes documentation

Most pods are configured using environment variables. They are a OS-agnostic standard that allows to change the configuration between deploys without changing any code in a very easy way. Sensitive information (e.g. passwords) is configured using Kubernetes secrets.

Working with deployments

A Deployment controller provides declarative updates for Pods and ReplicaSets, according to the desired state described in a Deployment object.

```
$ kubectl get deployments

NAME                    DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
admin-api               2         2         2            2           15m
apigw                   4         4         4            4           15m
authserver              3         3         3            3           15m
[...]
platform-health         2         2         2            2           14m
portal-backend          2         2         2            2           14m
portal-ui               2         2         2            2           14m
nginx-ingress                2         2         2            2           15m
```

The number of replicas depends on how many of them you specified in your deployment config.

ℹ️ You can get more information about working with deployments in the Kubernetes documentation.

Working with statefulsets

A Statefulset controller provides declarative updates for Pods that needs to maintain a state, for example using a persistent volume. Also, this controller provides guarantees about the ordering and uniqueness of these Pods.

The number of replicas depends on how many of them you specified in your statefulset configuration. However, not all of them scale nicely so it is really important to understand them well to be able to scale them safely.

```
$ kubectl get statefulsets -n baikal-$CORE

NAME                        READY   AGE
grafana                     1/1     8h
prometheus                  1/1     8h
prometheus-exporter-core    1/1     8h
prometheus-exporter-system  1/1     8h
redis                       1/1     8h
```

```
$ kubectl get statefulsets -n baikal-system

NAME                   READY   AGE
alertmanager           1/1     11h
elasticsearch          3/3     11h
fluentd-aggregator     5/5     11h
prometheus             2/2     11h
thanos-store-gateway   1/1     11h
```

```
$ kubectl get statefulsets -n baikal-state
NAME         READY   AGE
kafka        3/3     153m
prometheus   1/1     153m
zookeeper    3/3     153m
```

You have to keep in mind that:

- **grafana** uses a local database (sqlite), so you need to keep it to 1.
- **kafka** clusters use an affinity rule to guarantee that only one Kafka broker runs on each node in the "kafka" agent pool so you cannot exceed the number of nodes. It must be equal or greater than the Kafka replication factor, that is set to 3.
- **prometheus** stores the same information in all the available replicas, so it is recommended to keep it to 2 for HA reasons.
- **prometheus-exporter** only supports 1 replica.
- **redis** is used as a cache and doesn't form a cluster, so you need to keep it to 1.
- **zookeeper** ensembles also use a quorum protocol that needs an odd number of nodes. Three nodes is the minimum to have HA, so 3 nodes in preproduction and 5 in production environments is a good choice. Scaling Zookeepers ensembles over 7 nodes is proven to not have any benefit or even behave worst.
- **alertmanager** scales well adding or removing nodes.
- **elasticsearch** scales well adding or removing nodes.
- **fluentd-aggregator** scales well adding or removing nodes.
- **thanos-store-gateway** only supports 1 replica.

Once you are sure about it, use `kubectl` to scale the statefulset, for example:

```
$ kubectl scale statefulsets elasticsearch --replicas=5 -n baikal-$CORE
```

ℹ️ You can get more information about working with statefulsets in the Kubernetes documentation.

Working with nodes

The nodes are the virtual machines that run the Kernel Platform. There are two types of nodes in any Kubernetes cluster:

- Master nodes: host the control plane aspects of the cluster. They are managed by Azure AKS and they are not visible in Azure Portal.
- Compute nodes: nodes which are responsible for executing workloads for the platform services. There are different types of compute nodes. See "Computing" in the Cloud Infrastructure for more information about available types of nodes (also known as node pools or agent pools).

If there is an issue in the cluster, the first thing you should review is the status of the nodes.

```
$ kubectl get nodes

NAME                              STATUS   ROLES    AGE   VERSION
aks-system-22922262-vmss000000    Ready    agent   10h    v1.18.8
aks-system-22922262-vmss000001    Ready    agent   10h    v1.18.8
aks-system-22922262-vmss000002    Ready    agent   10h    v1.18.8
aks-common-28869154-vmss000000    Ready    agent   10h    v1.18.8
aks-common-28869154-vmss000001    Ready    agent   10h    v1.18.8
aks-common-28869154-vmss000002    Ready    agent   10h    v1.18.8
aks-common-28869154-vmss000003    Ready    agent   10h    v1.18.8
aks-common-28869154-vmss000004    Ready    agent   10h    v1.18.8
aks-drivers-28869154-vmss000000   Ready    agent   10h    v1.18.8
aks-genm-28869154-vmss000000      Ready    agent   10h    v1.18.8
aks-genmlow-28869154-vmss000000   Ready    agent   10h    v1.18.8
```

The normal status for a node is "Ready", that means that the node is up and a healthy member of the Kubernetes cluster. Nodes can become "NotReady" for different reasons when something is not right. In this case, the first step is to describe the affected node and check the "Conditions" and "Events" to determine what could be wrong:

```
$ kubectl describe node aks-common-28869154-vmss000000

...
Conditions:
  Type              Status
  ----              ------
  OutOfDisk         False
  MemoryPressure    False
  DiskPressure      False
  Ready             True
```

With `kubectl top nodes` you can get a quick overview of each node CPU and memory usage. If it is not enough and you need more details about the usage of resources, go to the Grafana dashboards.

If a node is misbehaving the recommended steps are:

1. Drain the Kubernetes node. This way, Kubernetes gives the pods running on that node a chance to stop in an orderly way and stops scheduling new pods on it. This step is not mandatory, but it is recommended.
2. Terminate the node. It is always safe to terminate one node at a time, waiting until it joins the cluster. Terminating many nodes at a time can affect the quorum of services that need to form a cluster (zookeeper, kafka, elasticsearch). so it is not recommended if the node you want to terminate contains these kind of pods.

Nodes that are cordoned appear with the status "SchedulingDisabled":

```
k8s-common-40654012-vmss000000      Ready,SchedulingDisabled   agent    ...
```

Filtering nodes with `kubectl` is very handy. For example, you can filter nodes to get those in a specific agent pool:

```
$ kubectl get nodes -l 'workload in (common)'
```

Or to get only those in a specific availability zone (fault domain)

```
$ kubectl get nodes -l 'failure-domain.beta.kubernetes.io/zone in (2)'
```

You can find information about the nodes usage in Grafana. Also, describing the nodes gives you information about how Kubernetes allocated resources on it:

```
$ kubectl describe node k8s-common-40654012-vmss000000

Non-terminated Pods:         (7 in total)
  Namespace                  Name                                    CPU Requests  CPU Limits   Memory Request
s  Memory Limits   AGE
  ---------                  ----                                    ------------  ----------   --------------
-  -------------   ---
  baikal-system              elasticsearch-es-http-0                 725m (18%)    1600m (40%)  7232Mi (63%)
   14464Mi (126%)  22m
  baikal-system              fluent-bit-6zf9p                        50m (1%)      100m (2%)    256Mi (2%)
   512Mi (4%)      20m
  baikal-system              node-exporter-tj4h2                     10m (0%)      50m (1%)     24Mi (0%)
   32Mi (0%)       20m
  kube-system                calico-node-vvw9j                       250m (6%)     0 (0%)       0 (0%)
   0 (0%)          20m
  kube-system                kube-proxy-gk5m7                        100m (2%)     0 (0%)       0 (0%)
   0 (0%)          20m
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource           Requests      Limits
  --------           --------      ------
  cpu                1235m (31%)   1950m (49%)
  memory             7612Mi (66%)  15208Mi (132%)
  ephemeral-storage  0 (0%)        0 (0%)
```

The CPU and memory limits can be over 100%. But the CPU and **memory requests** can't. This means that the resources requested by the pods also establish a limit even if they don't use the requested resources. If all nodes in an agent pool are full, new pods will wait in a "Pending" status until the cluster autoscaler adds a new node to the agent pool.

Autoscaling groups

Compute nodes belong to an a virtual machine scale set (VMSS). Each agent pool corresponds to one VMSS. Belonging to a VMSS means that, when a node is terminated for any reason, another one will be automatically created.

Horizontal scaling a component

Scaling deployments is easy using the `kubectl scale` command. It enables you to scale one or more replicated services either up or down to the desired number of replicas. For example, you might want to scale the number of replicas of the apigw deployment to 6:

```
$ kubectl scale deployment apigw --replicas=6 -n baikal-$CORE
```

Deployments contain stateless loads so they can safely scaled up and down. The only exceptions are those deployments that can only have one replica (review the list in "Working with deployments").

The platform supports Horizontal Pod Autoscalers. They have been included in some relevant services, to autoscale the number of replicas based the CPU/memory usage of a pod.

```
$ kubectl get hpa -n baikal-$CORE

NAME                    REFERENCE                          TARGETS     MINPODS   MAXPODS   REPLICAS   AGE
apigw                   Deployment/apigw                   1% / 80%    1         3         1          38m
authserver              Deployment/authserver              1% / 80%    1         3         1          38m
```

```
consent-events-manager   Deployment/consent-events-manager   6% / 80%    1         3         1          38m
events-provisioner       Deployment/events-provisioner       6% / 80%    1         3         1          38m
notifications-api        Deployment/notifications-api        4% / 80%    1         3         1          38m
notifications-retrier    Deployment/notifications-retrier    33% / 80%   1         3         1          38m
nginx-ingress                  Deployment/nginx-ingress                  6% / 80%    1         3         1
     38m
```

MINPODS values are set according to your deployment config using service_replicas value for each service. On the other hand, **MAXPODS** values are stablished as three times the value of MINPODS. **TARGET** is defined as a CPU utilization threshold. According to these policies, each service will scale up/down when needed, based on their own usage metrics. Support for additional custom metrics (e.g. latencies) is in the roadmap, to be added in future releases.

This feature is very powerful in combination with the cluster autoscaler, because once the pods created by the HPA don't fit in the available compute nodes, the cluster autoscaler will automatically add new nodes to the cluster.

Creating and deleting users

See Using Kubernetes Role Based Access Control (RBAC) to secure the Kernel Platform in the Kernel Platform Community Support center and the information about RBAC in the security section.

Remember that distributing the "root" kubeconfig file it is STRONGLY DISCOURAGED for production environments. For security reasons, nobody should use the "root" Kubernetes file and it should never be distributed. Instead, a different kubeconfig file must be generated for each user that needs to access the cluster.

Jobs

There are some scheduled jobs that run in the Kernel Platform. You can check them with `kubectl get jobs` :

```
$ kubectl get jobs -n baikal-$CORE

NAME                            COMPLETIONS   DURATION   AGEç
admin-api-provision             1/1           18m        8h
algorithm-manager-provision     1/1           23m        8h
apigw-provision                 1/1           9m14s      8h
apigw-provision-plugins            1/1           9m59s      8h
authserver-db-creation          1/1           59s        8h
authserver-migration            1/1           63s        8h
authserver-provision            1/1           12m        8h
core-essential-provision        1/1           16m        8h
data-api-provision              1/1           18m        8h
gdpr-provision                  1/1           17m        8h
global-portal-provision         1/1           18m        8h
grafana-provision               1/1           9m11s      8h
kafka-provision                 1/1           11m        8h
notifications-api-provision     1/1           19m        8h
platform-health-provision       1/1           18m        8h
portal-backend-provision        1/1           18m        8h
postgres-provision-core         1/1           47s        8h
postgres-provision-low-latency  1/1           36s        8h
user-events-manager-provision   1/1           19m        8h
users-metric-migrator           1/1           23m        8h
```

Most of them are provisioning jobs that create all the required entities during the installation (applications, APIs, etc). However, there are three special jobs that perform cleanup tasks:

- **authserver-purge**, **authserver-purge-authholders** and **authserver-purge-authentications** that run on a regular basis to remove unneeded entries from the authserver database.

The cleanup tasks are not critical for the service. Everything will keep working if one execution is skipped for some reason.

Moreover, there is another jobs that is in charge of perform periodic Zookeeper backup named **zookeeper-backup-XXX**. The Kernel Platform keeps working even if this job is not working but Zookeeper wouldn't be being backed up.

Horizontal scaling the infrastructure

It is possible to add and remove nodes to the different agent pools in the Kubernetes cluster.

Adding nodes to a running cluster is a safe operation. However, bear in mind that removing nodes can result in statefulsets not working properly. The reason is that some agent pools are dedicated to stateful services that need to form a cluster. That is the case of the following agent pools:

- **kafka**. The number of brokers in the kafka cluster can not be lower than the Kafka replication factor setting, that is set to 3. The kafka cluster needs to be rebalanced after adding new nodes.

Other services that run in the "common" agent pool but also need to form a cluster are:

- **zookeeper**. A zookeeper ensemble uses a quorum protocol that needs an odd number of nodes. Three nodes is the minimum to have HA, so 3 nodes in preproduction and 5 in production environments is a good choice.

Cluster autoscaler

The cluster-autoscaler is managed by Azure as part of the AKS control plane.

It is intended to automatically adjust the Kubernetes cluster size when one of this conditions are met:

- Scale out: there are pending pods that do not fit in the cluster due to insufficient available resources, but could fit if new compute nodes where added.
- Scale in: there are nodes in the cluster that have been underutilized for an extended period of time and their pods can be placed on other existing nodes.

This is the reason why it is not needed to configure the number of nodes in your deployment config. The cluster autoscaler will take care of everything to cut your cloud costs to the minimum.

ℹ️ You can find more information about the cluster autoscaler in the official Azure docs.

Vertical scaling the infrastructure

The process is similar to the horizontal scaling. You need to tune the `type` properties in the infrastructure section of your deployment config file. Also, you can tune the algorithm memory and cpu usage (1000 units = 1 vCPU).

```
infrastructure:
  compute:
    system_nodes:
      type: "Standard_F2s_v2"
    common_nodes:
      type: "Standard_DS3_v2"

#
# Infrastructure configuration for the Local Processing Environment where the algorithms run.
#
infrastructure_processing:
  compute:
    drivers:
      type: "Standard_D8s_v3"
      priority: "Regular"
    driversl:
```

```
      type: "Standard_D16s_v3"
      priority: "Regular"
    genxs:
      type: "Standard_D2s_v3"
      priority: "Regular"
    genxsspot:
      type: "Standard_D2s_v3"
      # priority: "Spot"  # TODO: Stop using Spot because Azure fails to create Spot pools with 0 instances. We
MUST find another solution before going to production.
      priority: "Regular"
    gens:
      type: "Standard_D4s_v3"
      priority: "Regular"
    gensspot:
      type: "Standard_D4s_v3"
      # priority: "Spot"  # TODO: Stop using Spot because Azure fails to create Spot pools with 0 instances. We
MUST find another solution before going to production.
      priority: "Regular"
    genm:
      type: "Standard_D8s_v3"
      priority: "Regular"
    genmspot:
      type: "Standard_D8s_v3"
      # priority: "Spot"  # TODO: Stop using Spot because Azure fails to create Spot pools with 0 instances. We
MUST find another solution before going to production.
      priority: "Regular"
    genl:
      type: "Standard_D16s_v3"
      priority: "Regular"
    genxl:
      type: "Standard_D32s_v3"
      priority: "Regular"
    genxlspot:
      type: "Standard_D32s_v3"
      # priority: "Spot"  # TODO: Stop using Spot because Azure fails to create Spot pools with 0 instances. We
MUST find another solution before going to production.
      priority: "Regular"
    gen2xl:
      type: "Standard_D64s_v3"
      priority: "Regular"
    gen2xlspot:
      type: "Standard_D64s_v3"
      # priority: "Spot"  # TODO: Stop using Spot because Azure fails to create Spot pools with 0 instances. We
MUST find another solution before going to production.
      priority: "Regular"

 algorithm_tiers:
   general-micro:
     pool_name: "genxs"
     algorithm_resources_percent: 100
     extra_algorithm_async_tasks: -1
     min_algorithm_executors: 1
     max_algorithm_executors: 5
     max_global_executors: 100
     driver_workload: "drivers"
     driver_resources_percent: 10
     enabled: True
   general-micro-spot:
     pool_name: "genxsspot"
     algorithm_resources_percent: 100
     extra_algorithm_async_tasks: -1
     min_algorithm_executors: 1
     max_algorithm_executors: 5
     max_global_executors: 100
     driver_workload: "drivers"
     driver_resources_percent: 10
     enabled: True
   general-short:
     pool_name: "gens"
```

```
      algorithm_resources_percent: 100
      extra_algorithm_async_tasks: -1
      min_algorithm_executors: 1
      max_algorithm_executors: 5
      max_global_executors: 100
      driver_workload: "drivers"
      driver_resources_percent: 10
      enabled: True
    general-short-spot:
      pool_name: "gensspot"
      algorithm_resources_percent: 100
      extra_algorithm_async_tasks: -1
      min_algorithm_executors: 1
      max_algorithm_executors: 5
      max_global_executors: 100
      driver_workload: "drivers"
      driver_resources_percent: 10
      enabled: True
    general-medium:
      pool_name: "genm"
      algorithm_resources_percent: 100
      extra_algorithm_async_tasks: -1
      min_algorithm_executors: 1
      max_algorithm_executors: 20
      max_global_executors: 100
      driver_workload: "drivers"
      driver_resources_percent: 25
      enabled: True
    general-medium-spot:
      pool_name: "genmspot"
      algorithm_resources_percent: 100
      extra_algorithm_async_tasks: -1
      min_algorithm_executors: 1
      max_algorithm_executors: 20
      max_global_executors: 100
      driver_workload: "drivers"
      driver_resources_percent: 25
      enabled: True
    general-large:
      pool_name: "genl"
      algorithm_resources_percent: 100
      extra_algorithm_async_tasks: -2
      min_algorithm_executors: 1
      max_algorithm_executors: 10
      max_global_executors: 100
      driver_workload: "driversl"
      driver_resources_percent: 50
      enabled: True
    general-large-spot:
      pool_name: "genlspot"
      algorithm_resources_percent: 100
      extra_algorithm_async_tasks: -2
      min_algorithm_executors: 1
      max_algorithm_executors: 10
      max_global_executors: 100
      driver_workload: "driversl"
      driver_resources_percent: 50
      enabled: True
    general-xlarge:
      pool_name: "genxl"
      algorithm_resources_percent: 100
      extra_algorithm_async_tasks: -2
      min_algorithm_executors: 1
      max_algorithm_executors: 10
      max_global_executors: 100
      driver_workload: "driversl"
      driver_resources_percent: 50
      enabled: True
    general-xlarge-spot:
```

```
      pool_name: "genxlspot"
      algorithm_resources_percent: 100
      extra_algorithm_async_tasks: -2
      min_algorithm_executors: 1
      max_algorithm_executors: 10
      max_global_executors: 100
      driver_workload: "driversl"
      driver_resources_percent: 50
      enabled: True
    general-2xlarge:
      pool_name: "gen2xl"
      algorithm_resources_percent: 100
      extra_algorithm_async_tasks: -2
      min_algorithm_executors: 1
      max_algorithm_executors: 5
      max_global_executors: 100
      driver_workload: "driversl"
      driver_resources_percent: 50
      enabled: True
    general-2xlarge-spot:
      pool_name: "gen2xlspot"
      algorithm_resources_percent: 100
      extra_algorithm_async_tasks: -2
      min_algorithm_executors: 1
      max_algorithm_executors: 5
      max_global_executors: 100
      driver_workload: "driversl"
      driver_resources_percent: 50
      enabled: True
    io:
      pool_name: "genm"
      algorithm_resources_percent: 100
      extra_algorithm_async_tasks: -1
      min_algorithm_executors: 1
      max_algorithm_executors: 15
      max_global_executors: 100
      driver_workload: "drivers"
      driver_resources_percent: 50
      enabled: True

infrastructure_database:
  core:
    type: MO_Gen5_4
    size_gb: 512
```

In Azure, it's not possible to change the instance types for a running cluster. This means that changing the instance types in the deployment config has no effects on redeployments. But, since the introduction of parallel-upgrade procedure, it is possible to create a new cluster with different instance types.

⚠️ Do not use the Azure Portal to modify the cluster nodes. It is an error-prone and unsupported way to scale the cluster that could impact the Kernel Platform stability. Kubernetes must be aware of the changes done to the cluster and all changes must be kept in sync with the deployment config and Terraform status.

Kubernetes storage

Kubernetes uses persistent volumes. There are backed by Managed Disks in Azure as described in the "Storage" section of the cloud infrastructure. The Kernel Platform uses the StorageClass to describe different types of storages.

AKS comes with some storage classes. The Kernel Platform adds two new storage classes to support deployments in multiple availability zones that guarantee that pods and their associated volumes run in the same zone:

- `baikal-standard` : Uses Standard disks.
- `baikal-managed` : Uses Managed disks.

More info about StorageClass could be found at the Kubernetes official docs.

AKS managed services

AKS deploys in the `kube-system` namespace some services used by the Kernel Platform. These services are not managed by the Kernel Platform team and its correct behaviour is an Azure responsability.

aks-link

These pods creates a tunneled connection between AKS nodes (owned by the Kernel Platform) and the Kubernetes control plane, that is fully managed by Azure (the Kernel Platform does not have access to its nodes).

In case of error connection between nodes and the control plane, Microsoft provides these troubleshouting guides.

azure-ip-mask-agent

This is a daemonset that modifies iptables rules on all AKS nodes, to mask the pod IP address with its node IP when sending traffic to public networks. This is a common pattern on Kubernetes clusters, and more information can be found in its documentation

azure-policy

Described in the azure documentation

coredns and coredns-autoscaler

Provides DNS based service discovery in AKS. More information in the Azure documentation.

csi-azure

It is the Azure implementation of the CSI (Container Storage Interface) standard, that integrates the Azure storage services with Kubernetes volumes. More information in the Azure documentation.

kube-proxy

The Kubernetes network proxy runs on each node. This reflects services as defined in the Kubernetes API on each node and can do simple TCP, UDP, and SCTP stream forwarding or round robin TCP, UDP, and SCTP forwarding across a set of backends. In AKS, it runs in iptables mode, that means that dinamically configure the iptables rules on each nodes to give access to the kubernetes services.

metrics-server

Starting from Kubernetes 1.8, resource usage metrics, such as container CPU and memory usage, are available in Kubernetes through the Metrics API.

Metrics Server is a cluster-wide aggregator of resource usage data that collects metrics from the Summary API, exposed by Kubelet on each node.

These metrics can be either accessed directly by user, for example by using `kubectl top command`, or used by a controller in the cluster, e.g. Horizontal Pod Autoscaler, to make decisions.

Kernel Platform Services

As the first grouping criteria, the services that compose the Kernel Platform are organized in 3 different layers:

- **Infrastructure services**. Services that are very tied to the infrastructure and could be reused by other products using this infrastructure.
- **System services**. Management services that are part of the Kernel Platform and could be potentially shared by several Kernel Platform deployments.
- **Core services**. All the other platform services that provide the end-user features.

All services that compose the Kernel Platform are run as Docker containers on the Kubernetes cluster. This helps us monitor and operate them in a consistent way. The exceptions are:

- the Kernel Platform database service, which is deployed as Postgres as a Service in the cloud provider to guarantee the reliability, performance and robustness of the solution.
- the Azure EventHubs used for dataAPI v2 as an scalable kafka-like queue, instead of using self-managed Kafka deployments.

System and Core services can be divided into different high-level groups:

- **NGINX Ingress Plus**: reverse proxy that handles all incoming HTTP/HTTPS requests.
- **Access Services**: providing OAuth-based access security, as well as traffic routing to internal and external services.
- **Management Services**: providing the tools to correctly operate the platform (logs, stats, alarms, etc).
- **State Kafka Services**: providing the internal backbone for services intercommunication within the Kernel Platform.
- **Other Core Services**: providing miscellaneous functionalities, such as administration interfaces, auditing, internationalization, transaction control, transient storage, etc.
- **Data Control layer**: controls who, how and when data is accessed. It interacts with the required components in order to manage the execution of the Data-IO plane as well as setting up of ddata topics at Kafka or EventHubs.
- **Data-IO layer**: is the one actually getting, filtering and preparing the data for reads and writes. On one side it accesses the Azure Blob Storage, and on the other side, it populates topics at Kafka or EventHubs.
- **EventHubs**: Azure service providing an event queue scalable implementation. It is used via its Apache Kafka interface. This service is used for all reads/writes via the new version 2 of the Data API. It is also used for all data reads/writes from algorithms that use the new version 0.5+ of the Kernel Platform SDK. Standard EventHubs is used for regular data access while Dedicated EventHubs clusters are set-up and torn-down automatically for larger data accesses.
- **Databricks**: cloud service for the execution of Spark processes. This is the preferred environment for running Kernel Platform Algorithms, using SDK v0.5+.
- **Processing Environment v1**: Transient execution environment for Spark algorithms running within the Kernel Platform Kubernetes cluster. This environment will be deprecated in favor of Databricks.
- **Notification System**: providing the possibility to subscribe to events and then resiliently notify them.
- **Consents Sync System**: providing the mechanism for the synchronization of consents between the OB and the platform.
- **Events Managers**: service that listen platform events and execute derived actions.
- **Portal**: our portal for OB administrators.
- **Adapters**: OB-specific processes in charge of adapting from Kernel Platform standard APIs to OB-specific representations and access models.
- **Legacy**: systems not to be evolved that will be decommissioned soon.

Infrastructure services

aad-pod-identity

This service allows pods in the Kubernetes cluster to obtain permissions to perform operations against the Azure API. Permissions are provided via Managed Identities.

cert-manager

`cert-manager` is a native Kubernetes certificate management controller. It can help with issuing certificates from a variety of sources, such as Let's Encrypt. It will ensure certificates are valid and up to date, and attempt to renew certificates before expiry.

For more info about cert-manager: https://docs.cert-manager.io

In the Kernel Platform, we are using it to issue a wildcard certificate signed by Let's Encrypt to be used by any service exposing TLS endpoints to the Internet such as NGINX Plus Ingress.

The Kernel Platform creates a cert-manager Issuer per tenant named `tenant-<tenant>-letsencrypt` that is in charge of managing certificates issuing with Let's Encrypt. You can get the Issuer status executing:

```
$ kubectl describe issuer tenant-<tenant>-letsencrypt
```

The Kernel Platform issues only one wildcard certificate per tenant named `tenant-<tenant>-cert-tls` whose status can be gathered executing:

```
$ kubectl describe certificate tenant-<tenant>-cert-tls
```

Once cert-manager renews a certificate, it is stored on a secret named `tenant-<tenant>-cert-tls`, and it is automatically applied to NGINX Plus Ingress.

If you need to force the certificate renewal you can simply delete the secret where the certificate is stored, and `cert-manager` will issue a new certificate:

```
$ kubectl delete secret tenant-<tenant>-cert-tls
```

⚠️ the `tenant-<tenant>-letsencrypt` Issuers and the `tenant-<tenant>-cert-tls` certificates should never be deleted.

keda

KEDA stands for Kubernetes-based Event Driven Autoscaler. It is built to be able to activate a Kubernetes deployment (i.e. no pods to a single pod) and subsequently to more pods based on events from various event sources. KEDA is a single-purpose and lightweight component that can be added into any Kubernetes cluster. KEDA works alongside standard Kubernetes components like the Horizontal Pod Autoscaler and can extend functionality without overwriting or duplication. With KEDA is possible to explicitly map the apps that will use event-driven scale, with other apps continuing to function. This makes KEDA a flexible and safe option to run alongside any number of any other Kubernetes applications or frameworks.

velero

Velero is an open source tool integrated in the Kernel Platform that allows us to make snapshots of the Kubernetes cluster persistent volumes and other Kubernetes resources.

Cluster Database

This is a PostgreSQL database used by some of the Kernel services to generate views. It is important to note that the information saved in this database is only a cache, and the real source of truth is in other system (e.g. Kafka).

The cluster database uses Azure Database for PostgreSQL - Flexible Server version 11. This means that the database is deployed as a cloud service in Azure instead of running within the Kubernetes cluster.

Database configuration is defined in the deployment config:

```
infrastructure_database:
  cluster:
    type: GP_Standard_D2s_v3
    version: "11"
    size_gb: 32
```

It is important to notice that you can not reduce the database size (`size_gb`) once it is created.

The databases servers are created in the common infrastructure resource group (resource groups descriptions). You can manage the database service through the Azure Portal, including monitoring and maintenance operations.

⚠️ Both instances of the database contain personal information. They should be accessed only by authorized employees.

To access to an Azure Database for PostgreSQL instance, you have to launch a dedicated pod in the Kubernetes cluster, acting as a database gateway.

```
kubectl run --restart=Never --image alpine --port 5432 -n baikal-system databasegw -- sh -c "apk --no-cache add socat && socat TCP-LISTEN:5432,fork TCP:postgres-cluster:5432"
```

Once the pod is running, you can access as usual with a port forward to the gateway pod and a local connection to the port exposed:

```
kubectl port-forward -n baikal-system databasegw 5432
```

After accessing the database instance, use the following command to delete the pod:

```
kubectl delete pod -n baikal-system databasegw
```

Database tier and max connections

Both Azure General Purpose and Memory Optimized databases are supported. Memory Optimized instances are recommended for production environments.

Even when most of the Kernel services use a connection pool to access to the database, the different Kernel services still use around 80 database connections. Take it into account because the database tier defines the maximum number of connections allowed. Bear also in mind that Azure requires five additional connections to monitor the database.

State services

The state uses its own Kubernetes clusters to deploy its services, because their lifecycle is different from the rest of Kernel Platform services.

Some of the infra services are also deployed at this state Kubernetes cluster:

- aad-pod-identity
- black-box-exporter
- fluent-bit
- kube-events-logger
- kube-state-metrics
- node-exporter
- prometheus
- reloader
- velero

burrow

Burrow is a monitoring companion for Apache Kafka that provides consumer lag checking as a service without the need for specifying thresholds. It monitors committed offsets for all consumers and calculates the status of those consumers on demand. Those metrics are exposed with an exporter running in the same pod. This way Prometheus can scrape the lag metrics. There is a panel available in the Kafka dashboard in Grafana.

external-dns

The external-dns operator makes kubernetes resources discoverable via public DNS services. It is used in the Kernel Platform to discover the private IPs of State services like Kafka, Zookeeper or Prometheus from the services Kubernetes cluster, using the Azure Private DNS service.

kafka

Kafka is the stateful service we use for internal communication among the services in the Kernel Platform.

It uses external volumes as its persistent storage.

Common troubleshooting include checking if Kafka topics have lag.

zookeeper

Zookeeper is a stateful service needed by Kafka to elect a controller, manage the Kafka cluster membership, the topic configurations, etc. Zookeeper must form a cluster (ensemble) and an odd number of nodes is needed to reach a consensus. Data stored in Zookeeper is critical and is automatically backed up to the object storage. The recommended number of nodes is 5 for production environments, so you can lose 2 replicas with no impact in the service availability.

Each zookeeper pod uses a 10GiB volume to store its data (/var/lib/zookeeper/data) and logs (/var/lib/zookeeper/logs). You can use the Kubernetes Storage dashboard in Grafana to check the used space. Remember that, if you lose the data stored in zookeeper, you also lose the Kafka cluster.

You can find more info about Zookeeper in the official administrator's guide.

Sometimes it is useful to know if a node in the Zookeeper ensemble is healthy, as well as other stats, such as if a specific node is the leader. You can connect to each node and run the following administrative commands:

```
kubectl exec -it zookeeper-1 -- bash

$ echo ruok | nc localhost 2181
imok

$ echo stat | nc localhost 2181
Zookeeper version: 3.4.10-39d3a4f269333c922ed3db283be479f9deacaa0f, built on 03/23/2017 10:13 GMT
Clients:
 /100.107.128.8:42818[1](queued=0,recved=14986,sent=14986)
 /100.111.0.7:52784[1](queued=0,recved=2993,sent=2993)
 /127.0.0.1:41322[0](queued=0,recved=1,sent=0)
 /100.117.0.2:53962[1](queued=0,recved=30246,sent=30299)

Latency min/avg/max: 0/0/287
Received: 193576
Sent: 193628
Connections: 4
Outstanding: 0
Zxid: 0x10000311f
Mode: follower
Node count: 2060
```

The mode ("leader" or "follower") lets us know the role of a specific node. There is only one leader in a Zookeeper ensemble and a node must always be in one of those two states. Since the release/3.10, a Kubernetes probe detects misbehaving nodes (for example, nodes that are not leaders nor followers) and automatically restarts their pods.

zookeeper-ui

This is a UI to browse data stored in Zookeeper using ZooNavigator. This UI is not publicly exposed so you need to run `kubectl port-forward services/zookeeper-ui 8000` and then open `http://localhost:8000/` in a browser.

⚠️ Be very careful because changing any of the Zookeeper zNodes could lead to the loss of the Kafka cluster.

Database

PostgreSQL is the core relational database used by some services in the Kernel Platform. The Kernel Platform uses Azure Database for PostgreSQL Single Server version 11. This means that the database is deployed as a cloud service in Azure instead of running within the Kubernetes cluster.

There are one server of Azure Database for PostgreSQL for each state creation:

- Server for the **core services** that needs a database as source of truth (currently only Authserver with all authentication data)

During upgrades, more databases are created. These databases are used to replicate the original databases data and, in case of rollback, recover the original state and data. The databases that are used by the Kernel Platform in a specified moment is marked with the `status: active` tag.

Database configuration is defined in the deployment config:

```
infrastructure_database:
  core:
    type: MO_Gen5_4
    size_gb: 512
```

It is important to notice that you can not reduce the database size (`size_gb`) once it is created.

The database server is created in the state resource group (resource groups descriptions). You can manage the database service through the Azure Portal, including monitoring and maintenance operations.

⚠️ The instance of the database contain personal information. They should be accessed only by authorized employees.

To access a to an Azure Database for PostgreSQL instance, you have to launch a dedicated pod in the Kubernetes cluster, acting as a database gateway.

- Database gateway pod for the core database instance:
  ```
  kubectl run --restart=Never --image alpine --port 5432 -n baikal-system databasegw -- sh -c "apk --no-cache add socat && socat TCP-LISTEN:5432,fork TCP:postgres-core:5432"
  ```

Once the pod is running, you can access as usual with a port forward to the gateway pod and a local connection to the port exposed:

```
kubectl port-forward -n baikal-system databasegw 5432
```

After accessing the database instance, use the following command to deletethe pod:

```
kubectl delete pod -n baikal-system databasegw
```

Also, it is possible access to the database through the pgbouncer pod:

```
kubectl exec -it deploy/pgbouncer -n baikal-ob-env -- bash -c "export PGPASSWORD='xxxxxxxxx';psql --host baikal-ob-env-core.postgres.database.azure.com --port=5432 --username=authserver_ob_env@baikal-ob-env-core authserver"
```

Database tier and max connections

Both Azure General Purpose and Memory Optimized databases are supported. Memory Optimized instances are recommended for production environments.

Even when most of the Kernel Platform services use a connection pool to access to the database, the different Kernel Platform services still use around 80 database connections. Take it into account because the database tier defines the maximum number of connections allowed. Bear also in mind that Azure requires five additional connections to monitor the database.

System services

This section describes each system service. Remember that there is a specific section for:

- the System Metrics Subsystem

alertmanager

ℹ️ Check System Metrics Subsystem for more information about "`alertmanager`".

blackbox-exporter

The blackbox-exporter is a service that allows probing of endpoints over HTTP, HTTPS, DNS, TCP and ICMP.

The Kernel Platform is deployed along with an external service/endpoint that is able to check its health from an end to end perspective. These checks include requesting an OAuth access token and sending a request to an API in the Kernel Platform. The sanity test suite is used to validate that the Kernel Platform is healthy. The blackbox-exporter uses an HTTPS probe that periodically sends an HTTP request to the external endpoint to validate that the Kernel Platform is still healthy. Its metrics (result, latency, etc) are stored in Prometheus.

descheduler

Descheduling involves evicting pods based on specific policies so that the pods can be rescheduled onto more appropriate nodes.

Your cluster can benefit from descheduling and rescheduling already-running pods for various reasons:

- Nodes are under- or over-utilized.
- Pod and node affinity requirements, such as taints or labels, have changed and the original scheduling decisions are no longer appropriate for certain nodes.
  - Node failure requires pods to be moved.
  - New nodes are added to clusters.

elasticsearch

ElasticSearch is a stateful service that indexes the Kernel Platform logs so they can be used for analysis. It runs as a statefulset. Logs can use a lot of space in disk, so it is important to size the volume accordingly by modifying the following section of the deployment config:

```
infrastructure:
  storage:
    logs:
      size_gb: 2048
```

The retention time of the logs is 10 days by default, but can be configured in the deployment config. Remember that increasing this value means that logs will take more space on disk and queries against ElasticSearch could take longer to complete.

```
service_configs:
  elasticsearch:
    retention: 10
```

You should also define a password for the admin user in the deployment config:

```
service_configs:
```

```
  elasticsearch:
    admin_password: PASSWORD
```

To check the disk usage you can use both the Kubernetes Storage or the ElasticSearch dashboard in Grafana.

Since release/4.3 version, an external endpoint `rawlogs` will be available to integrate an external Kibana or include the local elasticsearch cluster as a remote server in an external elasticsearch cluster deployment. This endpoint will publish two different paths, one for each opened port, /api (9200) for the REST interface of elasticsearch, and /control (9300) for the nodes communication interface.

For the nodes communication interface in port 9300, it will be necessary provide the elasticsearch certificate to the client want to connect to this interface, you can get it with the below command:

```
kubectl get secret elasticsearch-es-transport-certs-public -o go-template='{{index .data "ca.crt" | base64decod
e}}' > baikal.ca.crt
```

⚠️ You should be really careful to balance the load to this external endpoint to avoid any saturation scenario where the local and remote will miss any log access in the platform.

grafana

ℹ️ Check System Metrics Subsystem for more information about " `grafana` ".

Eventhubs logging namespace

baikal-< cluster >-cluster-logging is the eventhubs namespace used to decouple communication between fluent-bit daemonset and fluentd-aggregator statefulset. This architecture allow set different capacities between producers and consumers of log records.

It is an Standard namespace eventhubs configured with auto-inflate throughput from 1 to 20 throughput units to index all the logs generated in the Kernel platform. By default it is provisioned with four topics for each log domain:

- `baikal.log.kubernetes` to store all the logs from kubernetes containers
- `baikal.log.kafka` to store all the logs from kafka clusteres deployed in the platform, service, ingestion and logging.
- `baikal.log.algorithms` to store all the logs from the algorithms execution.

fluent-bit

Fluent-bit is a daemonset that runs in all nodes, processing these logs and sending them to the logging Event Hubs namespace. Each logs domain is ingested in a different topic, and this logs can be customized in the logging section in the config.

By default, Kubernetes and Kafka logs (all the Kafka clusters deployed in the platform) are indexed in the ElasticSearch. There are 2 optional log sources in the platform disabled by default, OS logs and audit logs.

OS logs are gathered from the systemd service, a centralized solution for logging all kernel and userland processes. Be careful with this option because systemd logs are very verbose and generate a high number of documents to be indexed.

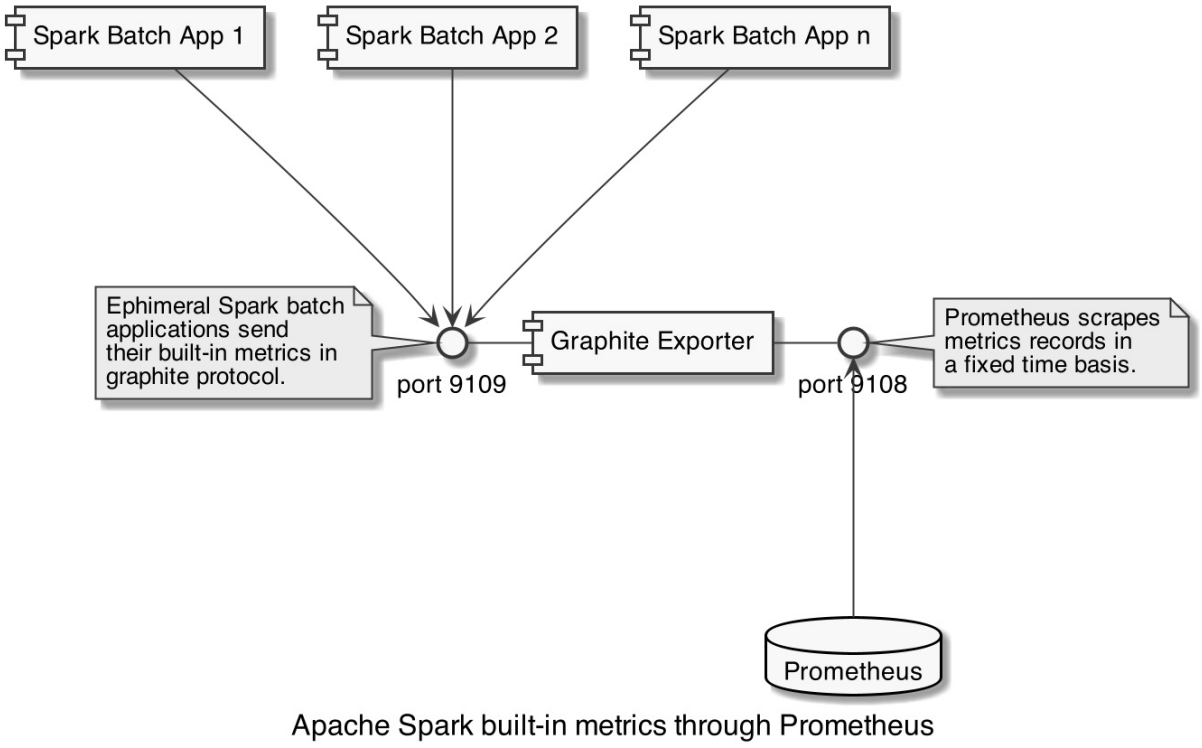
fluentd-aggregator

Stateful service that aggregates all logs coming from the Fluent-bit daemonset on every node. This aggregator consumes all the logs ingested in the logging eventhubs namespace and indexes them in ElasticSearch. It has a small data disk of 10GB that acts as a buffer to avoid losing data if something goes wrong (e.g. a network issue or a problem with the ElasticSearch cluster) while trying to index the logs.

It is safe to kill a specific fluentd-aggregator pod (`kubectl delete pod`) if it gets stuck for some reason, log records will not be missed.

graphite-exporter

A server that accepts built-in metrics generated by the Spark processes running in the processing environment via the Graphite protocol and exports them as Prometheus metrics. The pods run in the baikal-system namespace.



Apache Spark built-in metrics through Prometheus

kibana

Kibana is the service used in the Kernel Platform to access logs indexed in ElasticSearch. The first time you access Kibana, you need to create an index mapping against "baikal-core-*", using @timestamp as the temporal reference for logs. Remember that logs are stored in ElasticSearch for 10 days by default.

kube-events-logger

Process that watches Kubernetes events from all namespaces (i.e. kubectl get events) so they are indexed in ElasticSearch for further usage.

Due to the way it is implemented, watching events using kubectl with `-w --watch-only` flags), it is normal to see restarts in this pod.

kube-state-metrics

The kube-state-metrics is an official Kubernetes service that listens to the Kubernetes API and generates metrics about the state of the objects, such as deployments, nodes and pods.

kubed

Kubed is a Kubernetes Cluster Operator Daemon that keeps configmap and secret synchronized between namespaces.

kured

Kured (KUbernetes REboot Daemon) is a Kubernetes daemonset that performs safe automatic node reboots when the need to do so is indicated by the package management system of the underlying OS.

It is enabled by default. You can disable it in the deployment config if needed, setting `security.enable_kured: False`, but it is not recommended for security reasons.

Kured runs in all nodes, watching for the presence of a reboot sentinel (e.g. `/var/run/reboot-required`) in the node's filesystem. When it finds one, it cordons and drains cluster nodes in an orderly way before rebooting it, to minimize the chances of service loss. Once the node has been updated, it is uncordoned.

There is a metric `kured_reboot_required` and one alarm is fired when this metric is greater than zero for more than 24h. It means that a node needs to be rebooted but kured failed or was not allowed to do it. In this case, manually cordon and restart the node is the recommended solution.

Some pods are marked with a special label `node-reboot=block`. Even when the Kernel Platform is designed to be resilient if a node is restarted or lost, we prefer not to kill nodes where any of these pods are running:

- Spark drivers that run in the processing environment.
- Kafka and Zookeeper, that maintain status and are more fragile.

It uses a lock in the Kubernetes API server to ensure only one node reboots at a time.

node-exporter

The node-exporter is an official Prometheus exporter that gathers hardware and OS metrics exposed by the virtual machines that form the Kernel Platform.

prometheus

Check System Metrics Subsystem for more information about "`prometheus`".

prometheus-exporter

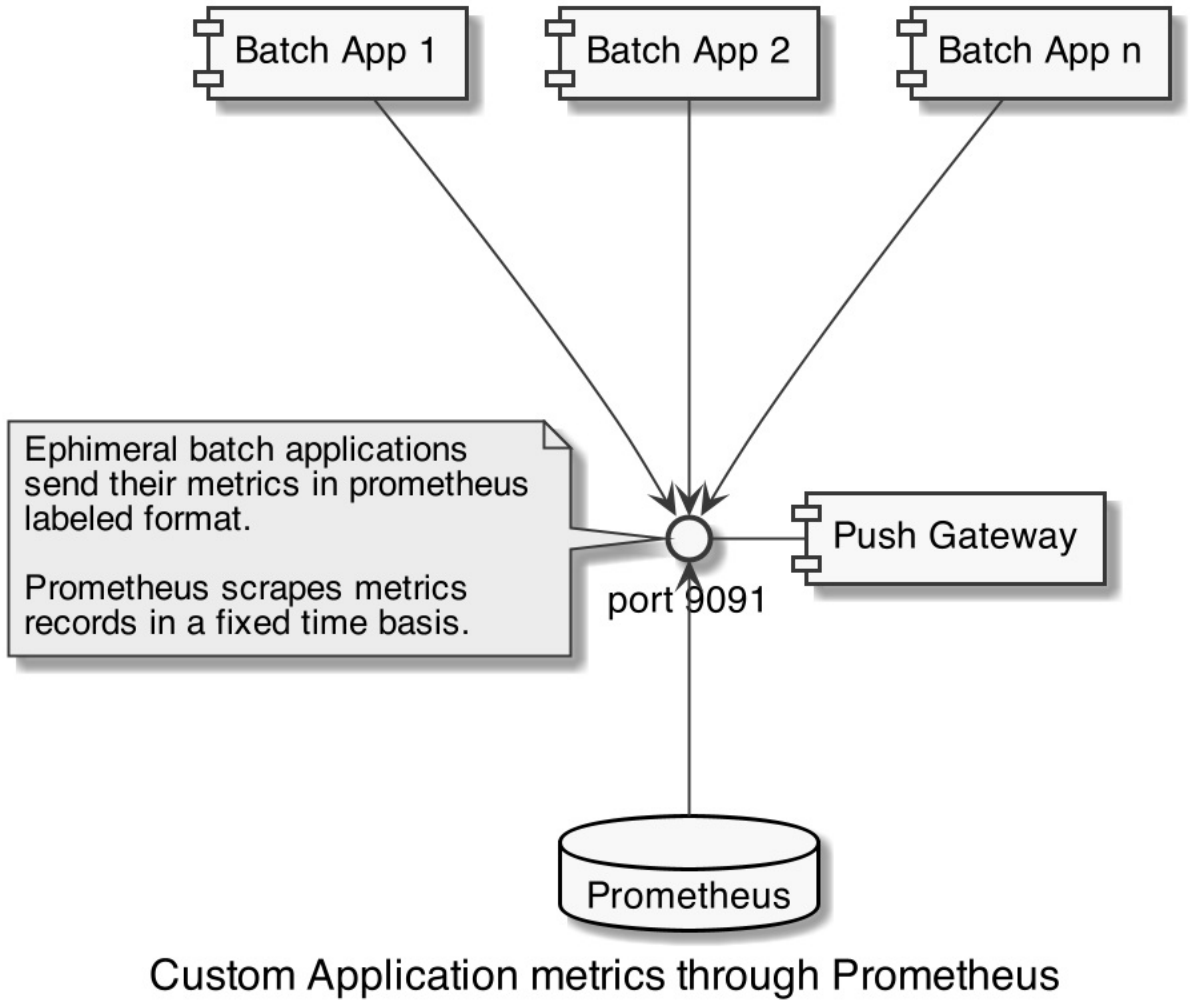
This is a couple of federated prometheus, one for core and other one for system, that gathers some metrics from the main prometheus at a low scraping rate and sends them to prometheus-kafka-receiver, that is configured as a remote_write endpoint. These metrics are those that need a longer retention time, such as the ones needed to generate the Kernel Platform KPIs.

prometheus-kafka-adapter

This service receives metrics from prometheus-exporter and sends them to a Kafka topic (.baikal.core.metrics). The metrics can be consumed for different purposes, such as auto-scaling the platform (not currently used) or generating the Kernel Platform KPIs. The metrics are considered a dataset in the Kernel Platform, and end up stored in metrics v2 dataset using the algorithm metrics-exporter, that is scheduled hourly by the algorithm-manager.

pushgateway

Check System Metrics Subsystem for more information about "`pushgateway`".



Custom Application metrics through Prometheus

reloader

The reloader operator monitor changes in config maps and secrets used by the replicasets with a specific annotations to do a rolling restart if they are modified.

thanos

Check System Metrics Subsystem for more information about "`thanos`".

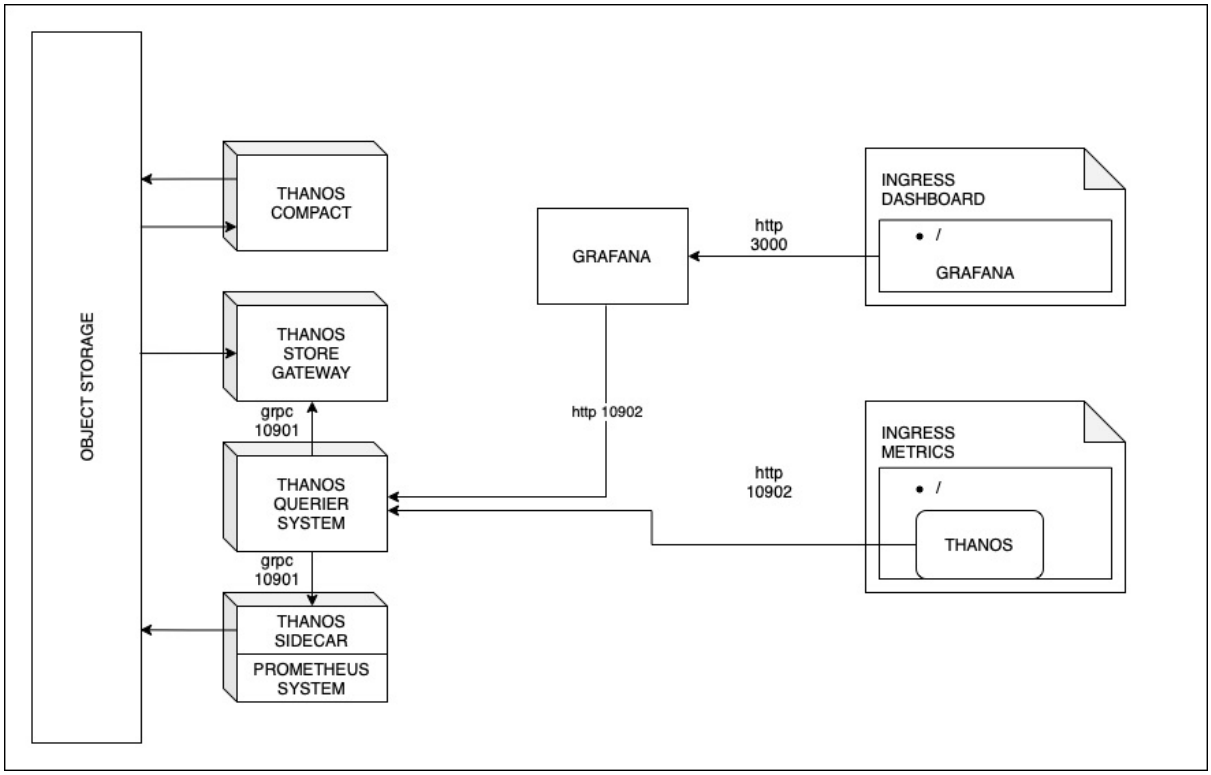
nginx plus ingress

NGINX Plus Ingress is the entry point for the APIs exposed in the Kernel Platform. This doesn't include the Kubernetes API. It sits behind the cloud load balancer service (Azure Load Balancer) and is in charge of:

- Add basic auth to some Kernel Platform management services (alertmanager, prometheus and kibana).
- Restrict access to the IP ranges defined in the `service_whitelist_source_ranges` configuration.
- Route incoming requests to the corresponding backend pod (mostly based on the host domain and request path).
- Implement WAF (Web Application Firewall) and L7 DoS (Denial of Service).

NGINX Plus Ingress uses the cert-manager service to obtain a Let's Encrypt certificate.

Metrics Subsystem



The metric subsystem is built on top of multiple services that extend across the span of the platform. This chapter describe its components located at the baikal-system namespace.

prometheus

The " `prometheus` " component is a system and service monitoring process. It collects metrics from discovered targets across the Kernel Platform at given intervals and can trigger alerts if some condition is observed to be true.

The system namespace variant discover and scrape samples from components (pods or endpoints) located at any namespace that match the following Kubernetes object annotation:

```
prometheus.io/scrape: true
```

If the local storage becomes corrupted for whatever reason, your best bet is to shut down Prometheus and remove the storage directory, that is mounted under `/prometheus` on the Prometheus container.

The recommended pattern for running Prometheus in HA mode is to run duplicated instances (same configuration, scraping the same targets independently). That means having at least 2 replicas running.

prometheus container

The " `prometheus` " container is responsible of discovering targets. It scrapes samples from them and ingest into new or existing timeseries locally.

It is a stateful component, as it temporally store data to a volume (for redundancy purposes) and each replica has different data. However it can be restarted and/or its data deleted at risk of lossing up to the configured block duration.

thanos-sidecar container

The " `thanos-sidecar` " container is a sidecar to the " `prometheus` " container that enhances it by exposing a gRPC StoreAPI and by uploading blocks to an object storage API (like Azure Blob Storage).

It is a stateless component.

thanos-querier

The " `thanos-querier` " container expose a gRPC StoreAPI and an HTTP Prometheus v1 API. It gathers the data needed to evaluate the query from underlying StoreAPIs, evaluates the query and returns the result.

It is a stateless component.

The configured StoreAPI sources are:

- system namespace -> prometheus/thanos-sidecar
- system namespace -> thanos-store-gateway

ingress

The " `thanos-querier` " component is exposed to the internet through a Kubernetes Ingress object and receives any request with the following prefix in their request path:

- /api/v1

Requests that also match other prefixes more restrictive will be redirected to their respective components. The main purpose for this route is to serve the PromQL queries from " `thanos-querier` " instead of individual " `prometheus` " replicas.

thanos-compact

The " `thanos-compact` " container is a component that applies the compaction procedure of the Prometheus 2.0 storage engine to block data stored in object storage APIs (like Azure Blob Storage). It also generate downsampled blocks from each raw block.

It is a stateful component, as it must be deployed as a singleton (against an exclusive label selector).

thanos-store-gateway

The " `thanos-store-gateway` " container expose a gRPC StoreAPI. It serves data blocks containing metrics stored in Azure Blob Storage.

It is a stateless component, however it consume local storage for sync purposes and benefits from persistence against increased startup times.

alerts-rules

The PrometheusRule custom resource definition (CRD) declaratively defines a desired Prometheus rule to be consumed by one or more Prometheus instances.

Alerts and recording rules can be saved and applied as YAML files and dynamically loaded without requiring any restart.

grafana

Grafana is an opensource service for analytics and monitoring. The Kernel Platform uses it to display metrics from Prometheus in several dashboards. It uses a sqlite3 database to store the dashboards, and redis to store the user sessions. Using sqlite3 means that only one replica of grafana can be running in the platform.

⚠ Changes to the official Kernel Platform dashboards (those with the "baikal" label) will be overridden on every deployment of the Kernel Platform infrastructure. ⚠ Grafana is not prepared to alert the Kernel Platform. The Kernel Platform interfaces are the Prometheus API and the alertmanager.

dashboards

The GrafanaDashboards custom resource definition (CRD) declaratively defines a desired Grafana Dashboards to be consumed by Grafana.

Dashboards can be saved and applied as YAML files (grafana-infra-dashboards, grafana-system-dashboards), grafana-core-dashboards) and dynamically loaded without requiring any restart.

alertmanager

The alertmanager is part of the Prometheus suite and is in charge of sending notifications (email,) when an alert goes off in Prometheus. It handles alerts sent by Prometheus server. It takes care of deduplicating, grouping, and routing them to the correct receiver integration by email. It also takes care of silencing and inhibition of alerts.

Notifications are sent to the `notifications_email` (defined in the deployment config) using an external global SMTP server administered by the Kernel Platform Team. It is important that the different teams that operate the platform are subscribed to the alerts.

In the Kernel Platform, Prometheus server is deployed with a full set of alerts to monitor all the critical functionalities currently delivered.

pushgateway

The Prometheus Pushgateway allow ephemeral and batch jobs that run in the processing environment to expose their metrics to Prometheus. Since these kinds of jobs may not exist long enough to be scraped, they instead push their metrics to a Pushgateway. The Pushgateway then exposes these metrics to Prometheus.

Cronjobs

apigw jwk keys rotation

There are two cronjobs to implement the jwk keys rotation lifecycle in apigw, one cronjob to expire the current key and generate a new one, another cronjob to do the safe removal of the expired key after 24h.

apigw-jwk-expire

This cronjob runs every Sunday at 2:05 am to begin the rotation of private keys to encrypt tokens. This job will create a new private keyset and will give the existing one a 1 hour expiration time. Note that this expiration is just for exposing the public key, the apigw will be able to validate tokens encrypted with the old one until this key is deleted, no matter is expired. See also the description of the next cronjob as they should work together to achive the rotation

apigw-jwk-rotate

This cronjob runs every Monday at 05:05 am (24 hours plus 3 hours of gap from the previous job execution time). This job will replace the old keys (already expired) with the new key. An access_token emitted with the previous job may have a maximum validity of 24 hours, we give 1 hour to update apigw internal decryptors cache, another hour to update authserver encryptors and 1 hour extra. This will allow all tokens to be decrypted without issues during the rotation.

authserver jwk keys rotation

There are two cronjobs to implement the jwk keys rotation lifecycle in authserver, one cronjob to expire the current key and generate a new one, another cronjob to do the safe removal of the expired key after 24h.

authserver-jwk-expire

This cronjob runs every Tuesday at 2:05 am to begin the rotation of private keys to encrypt tokens. This job will create a new private keyset and will give the existing one a 1 hour expiration time. Note that this expiration is just for exposing the public key, the authserver will be able to verify tokens signed with the old one until this key is deleted, no matter is expired. See also the description of the next cronjob as they should work together to achive the rotation.

authserver-jwk-expire

This cronjob runs every Wednesday at 05:05 am (24 hours plus 3 hours of gap from the previous job execution time). This job will replace the old keys (already expired) with the new key. An access_token/id_token emitted with the previous job may have a maximum validity of 24 hours, we give 1 hour to update apigw internal validators cache, another hour to update authserver signers and 1 hour extra. This will allow all tokens to be verified without issues during the rotation.

authserver purgers

The authserver stores in its database information related with authentication and refresh tokens. This information expires over time so it's needed a process to expire that information and also to clean the database of unused entries.

This is achieved using 3 Kubernetes cronjobs that are executed with a given periodicity.

- authserver-purge: It cleans expired refresh tokens, authorization codes (those not consumed) and device codes.
- authserver-purge-authholder: It cleans authentication holders (it contains authentication metadata) that are not referenced by any refresh token or cookie.

- authserver-purge-authentications: It cleans authentication information (external IDP data, identities and related stuff) that is not referenced by any authholder.

Descheduler

This cronjon is executed each ten minutes and its functionality involves evicting pods based on specific policies so that the pods can be rescheduled onto more appropriate nodes.

Kafka backup

This cronjob is executed on a daily basis to backup the Kafka zNodes to the "backups" storage account on the cloud. Note that this cronjob is launched in the core Kubernetes cluster.

notifications-purge

The notifications-api stores in its database information related with notifications processed by the system. It's needed a process to clean the database of expired entries with the retention time.

postgres-vacuum

This is a cluster database maintenance cronjob to vacuum those tables which exceed the dead rows threshold. Actually, this threshold is configured in 2000 dead rows.

thanos-compactor

This cronjob runs everyday at 00:00 and , among other things, it is responsible for compacting multiple blocks into one. This is a process, also done by Prometheus, to reduce the number of blocks and compact index indices. We compact an index because series usually live longer than the duration of the smallest blocks (2 hours).

Zookeeper backup

This cronjob is executed on a daily basis to backup the Zookeeper zNodes to the "backups" storage account on the cloud. Note that this cronjob is launched in the state Kubernetes cluster.

It leverages Minio as a S3 gateway to abstract the details of the underlying cloud. This means that all Kernel Platform services that use Minio see a S3-compatible interface (see Minio gateway for Azure for additional information).

It is possible to get the associated pod's status:

```
$ kubectl get pod -l job=zookeeper-backup

NAME                                READY   STATUS    RESTARTS   AGE
zookeeper-backup-1569484800-wc9r6   1/2     Running   0          3h28m
```

The pod has two containers:

1. Minio acting as a gateway between the Kernel Platform and the object storage
2. The backup process itself (zookeeper-backup), that uses burry and communicates with Minio via localhost to save the backup.

It is normal to see this job as Running with 1/2 containers. This is due to the fact that Minio keeps running until the cronjob is retriggered and the pod is replaced by a new one.

Minio runs as a sidecar container in the same pod. If something goes wrong between the Kernel Platform and the object storage (e.g. bad credentials, network issues), the best way to start looking for the issue are the "minio" logs:

```
$ kubectl logs zookeeper-backup-1569484800-wc9r6 -c minio
```

The logs of the zookeeper-backup process should show that the operation successfully completed:

```
$ kubectl logs zookeeper-backup-1569484800-wc9r6 -c zookeeper-backup

time="2019-09-26T08:00:24Z" level=info msg="Selected operation: BACKUP" func=main
2019/09/26 08:00:24 Connected to 100.68.147.162:2181
2019/09/26 08:00:24 Authenticated: id=72062961509859372, timeout=4000
2019/09/26 08:00:24 Re-submitting `0` credentials after reconnect
time="2019-09-26T08:00:24Z" level=info msg="Rewriting root" func=store
time="2019-09-26T08:00:31Z" level=info msg="Successfully stored borja-default-dev-backups/zookeeper-4.0/1569484
824 (539900 Bytes) in S3 compatible remote storage 127.0.0.1:9000" func=toremoteS3
time="2019-09-26T08:00:31Z" level=info msg="Operation successfully completed. The snapshot ID is: 1569484824" f
unc=main
```

:information_notice: See the "Backup" section to find extended information about the backup and how to run a restore procedure if needed.

Core services

This section describes each core service. Remember that there is a specific section for the services that form

- the Processing Environment
- the Data Control layer
- the Data-IO layer
- the Notifications System
- the Audit System
- the Consents Sync System
- the Access Services
- the Events Managers
- the Portal
- the Adapters

admin-api

The admin-api exposes the Kernel Platform administrative API. It is not expected to process a lot of requests, so having 2 replicas for HA should be enough. There is a Grafana Dashboard that shows metrics related to the admin-api.

privacy-manager

The micro-service is running behind apigw and admin-api, redirecting the requests to the OB (common-oauth adapter) or processing directly itself for consents, legal entities, purposes and pi-scopes.

operations-manager

The operations-manager exposes an HTTP API to get information on the progress of asynchronous provisioning operations. Many operations executed against the Kernel Platform are asynchronous and return an operation identifier (or operationId).

The operationId is added as a field in all of the logs that are emitted in any component of the Kernel Platform that is performing actions based on that operationId. This means it is an ideal filter for Kibana when searching for logs and to debug and understand how an operation is processed throughout the platform.

The operations-manager uses a multi-level cache, for operationId information. It keeps 2000 hot entries in memory, and uses a 1GB memory-mapped file for the cold entries. This means that information about operationIds don't disappear at a fixed rates, but instead depend on the load of the platform. However, they are generally kept around a lot of time, since 1GB is plenty of storage.

Since the operations-manager has a memory mapped file, is normal for it to use as much memory as it can, since the OS will map into memory as much of the file as possible.

Another thing the operations-manager does is to rollback those operations that failed due to an internal error of the platform. If there is a failed operation message, the operations-manager will look for the message that started that operation and check whether there is a rollback message defined. If that is the case, it will inject the message into the platform so the platofrm as a whole can revert to a clean state.

The rollback feature can be enabled or disabled through the `ROLLBACK_ENABLE` environment variable.

platform-health

The platform-health is an internal service used to verify the overall health of the Kernel Platform. There is a public endpoint, dynamically created the first time the Kernel Platform is deployed, that checks the Kernel Platform's health sending an end-to-end request to the service. This endpoint is shown when the installation finishes and is very useful for monitoring.

redis

Redis is a stateful service with only one replica, used as a cache service by several services. The service is not critical for the Kernel Platform availability, since all services are designed to work with a degraded performance, even if the cache is down.

It is limited to 2GiB memory and with a LRU policy that discards the least recently used keys if the memory limit is reached.

This is the redis databases list already in use:

- Database 0: the adapters store API responses that can be reused
- Database 1: the grafana stores its sessions
- Dabatase 2: the authserver stores clients, purposes, scopes, ...
- Database 3: the privacy-manager stores legal-entities, purposes, pi-scopes and consents.
- Database 4: the algorithm-manager stores algorithm spark pod statuses to improve algorithm statuses calculations

Access services

apigw

The API Gateway (apigw) is the service that authorizes requests to the Kernel Platform APIs. It is based on Kong Community Edition, with some additional plugins. It is in charge of:

- Validating that incoming requests come with a valid OAuth token. Tokens in the Kernel Platform are self-contained, so the apigw only needs to validate their signature and payload, with no need to access an external service to validate the token.
- Forwarding the request to the appropriate upstream container to process the request.
- The API Gateway controls the number of request per minute for a client, to avoid DDoS attacks or limit apply rate limits to some endpoints.

The apigw uses the PostgreSQL database named "apigw" to store its configuration. The connection to the database is done through pgbouncer, that acts as its connection pool.

The apigw is a stateless service. It is able to process around 500 requests per second per core and its deployment can be safely scaled to add or remove replicas when needed.

The apigw generates audit events in JSON format, that are saved to the node disk and sent to the kafka topic `<core>.baikal.core.events-apigw` using `filebeat`.

In the apigw, the client header buffer is configured to 4k to guarantee that most Kernel Platform tokens fit in Authorization Bearer header without opening additional buffers, it means, all upstreams integrated with Kernel Platform, at least, can receive headers with 4k size, which could be bigger if some mediation element in the architecture add some additional header. So it is really important ensure this requisite with all the thrid party platforms integrated with Kernel Platform to avoid complex integration issues.

authserver

The authserver is a service with two responsibilities:

- IdP Broker: it is able to select and integrate with the appropriate authenticator in the OB side.
- Authserver: after the user is authenticated, it is able to authorize the access to the Kernel Platform resources based on the issuance of temporary credentials.

The authserver uses two PostgreSQL databases (named "authserver") to store its information in different servers and support Zero Downtime platform upgrades:

1. Authentication and authorization data
2. Administrative data like client information, grants, redirect URIs, system scopes...

It is able to generate around 100 tokens per second per core and its deployment can be safely scaled to add or remove replicas when needed.

The authserver generates audit events in JSON format, that are saved to the node disk and sent to the kafka topics `<core>.baikal.core.events-authserver-idp-sessions` and `<core>.baikal.core.events-authserver-authorizations` using `filebeat`.

pgbouncer

PgBouncer is a lightweight connection pooler for PostgreSQL. We use it in transaction pooling mode where a server connection is assigned to a client only during a transaction. When PgBouncer notices that the transaction is over, the server will be put back into the pool. This pooler has Low memory requirements (2 kB per connection by default). This is because PgBouncer does not need to see full packets at once.

Currently, only the apigw connect to the database using pgbouncer because they do not have their own database pool. We take the control on how many connections are opened to database and we increase the performance (if we have many apigw instances due to a peak, more then 10 connections opened will reduce the db operations speed).

Microservices that adapte HTTP requests and response between standard Kernel Platform and OB formats, potentially requiring to mash-up different OB endpoints.

Some of the adapters are as well focused on integrating with OB access security mechanisms, such as 2waySSL or OAuth tokens.

filebeat

The apigw and the authserver generate audit events that are saved to the node local disk. Filebeat is a daemonset that reads these files and asynchronously sends the events to kafka. The local disk acts as a buffer, ensuring events are eventually sent to kafka even if there are problems connecting with it during a period of time (1 day). Also, filebeat commits the offsets only when an ack is received to minimize event losses.

logrotate

This service periodically compresses and deletes the old audit files that apigw and authserver writes to disk, to avoid disk space problems on nodes. As filebeat, is deployed as a daemonset to ensure all nodes have a logrotate service running on it.

Adapters

Microservices that adapt HTTP requests and response between standard Kernel Platform and OB formats, potentially requiring to mash-up different OB endpoints.

Some of the adapters are as well focused on integrating with OB access security mechanisms, such as 2waySSL or OAuth tokens.

Data Control layer

This layer controls who, how and when data is accessed. It interacts with the required components in order to manage the execution of the Data-IO plane as well as setting up data topics at Kafka or EventHubs. It is composed by the following microservices.

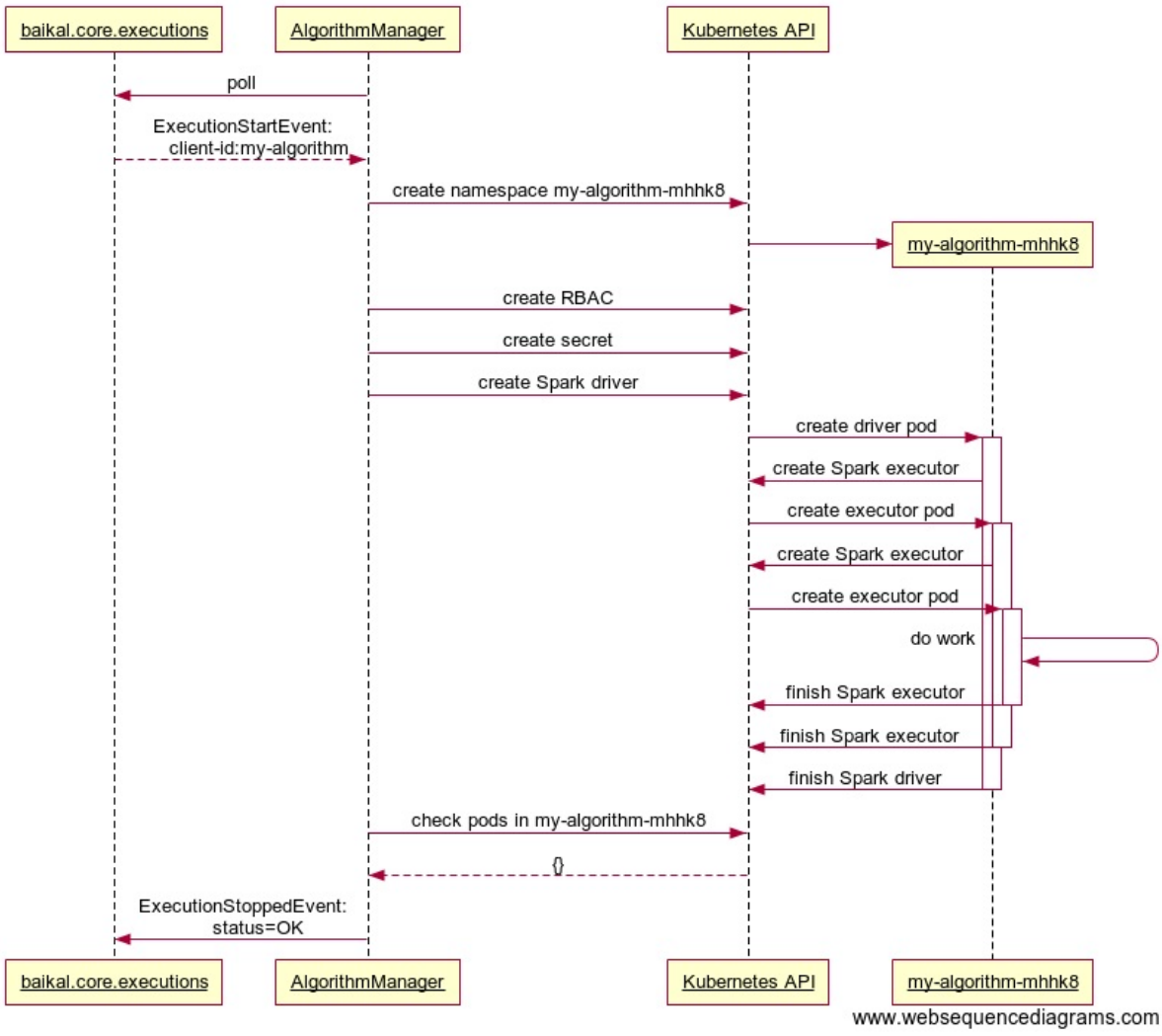
algorithm-manager

The algorithm manager allows managing the algorithms executed in the Kernel Platform. This service launches Spark jobs on the Kubernetes cluster. The algorithm manager responds to messages in the `baikal.core.executions` Kafka topic by triggering the corresponding jobs. It will then monitor the Spark jobs and update the status in that same Kafka topic. Monitoring of running algorithms happens by injecting messages into the `baikal.core.executions-inprogress` topic.

When launching an algorithm, the algorithm-manager interacts with the Kubernetes API to create and manage the lifecycle of several Kubernetes resources. These include:

- A namespace for the Spark driver and workers
- RBAC for that namespace (this includes a service account, a role and a role binding)
- A Kubernetes secret, to propagate sensitive information

It also interacts with the Kubernetes API through the use of Spark on Kubernetes in order to launch the Spark driver and worker pods in the `processing` agent pool.



Algorithms have an associated timeout, so if something goes wrong (e.g. the algorithm never finishes, or Kubernetes is not able to allocate resources for an execution) then the algorithm-manager will delete the associated Kubernetes resources once the timeout expires. It will also mark the execution as failed in the Kafka topic.

Additionally algorithm-manager expose an Algorithms API. This API allows to find driver's execution logs by execution-id using the ElasticSearch API. Additionally, we also can build algorithms using this API. To perform this action, the algorithm-manager uses the Azure Container Registry Tasks (also called ACR Tasks) service. These tasks allow to build a docker image from a Dockerfile along with the arguments required. You can have more info about an ACR task execution in the Azure Portal, in the Container Registry section.

However, since the Kubernetes resources for algorithms are deleted as soon as the run finishes, the logs can't generally be retrieved through `kubectl`. In order to access the logs, the operator must log into ElasticSearch and filter by `kubernetes.labels.spark-role` -> `driver` and `kubernetes.labels.execution-id` -> `${execution-id}`. The execution-id can be found in `<core>.baikal.core.executions` or the algorithm-manager logs and is obtained after launching an algorithm via the Algorithms API.

For example, if you want to get the log of when and how and algorithm was run, you can do a search for the "exec" text chain with aforementioned filters activated, and you will get something like the following:

```
exec /sbin/tini -s -- /opt/spark/bin/spark-submit --conf spark.driver.bindAddress=10.240.1.2 --deploy-mode client --properties-file /opt/spark/conf/spark.properties --class com.telefonica.baikal.metricsexporter.Main spark-internal 2020-11-18T09:00:00Z
```

If you need to re-execute that algorithm with the same arguments (note they are separated by blank spaces in the command log above), you can use the Algorithms API. Please, refer to https://developers.baikalplatform.com/docs/datasets/processing-data to see further information on how to run, stop and get information on algorithms execution in general.

data-api

Spark jobs will most likely want to read and/or write data to/from the Kernel Platform. They achieve this through the data-api. The data-api receives a read/write request and prepares a staging area for the data. The staging area consists of a data-io Spark job and an EventHubs. The data-api manages the lifecycle of the EventHub by listening on `<core>.baikal.core.executions` to determine when all consumers have stopped.

The data API is also exposed publicly to clients outside the Kernel Platform, in order to allow data reads and writes. For Data API v2, read and write request always create an EvenHubs topic to execute the operation. In order for clients to complete a read/write two more endpoints are available:

- `/status`: this endpoint will return the current status of a particular read or write (creating, populated, ready, finishing, finished, failed).
- `/setting`: provides the details to connect to the EventHubs topic for interactions via the Data API v2.
- `/end`: this allows the client to signal it has finished reading or writing the data.

For more details, please, take a look at the API reference documentation

The data-io is launched through the algorithm-manager and its lifecycle is managed through the algorithm-manager.

Data-IO layer

This layer is the one actually getting, filtering and preparing the data for reads and writes. On one side it accesses the Azure Blob Storage, and on the other side, it populates topics at Kafka or EventHubs. It is composed by several instances of the same Spark algorithm: the data-io. There is one instance of the data-io running to handle each read or write.

data-io

The data-io is a Spark job whose goal is to populate data into the EventHubs or save the data from the EventHubs into its final destination (in case of write requests). It runs in the processing nodes of the cluster and its lifecycle is managed by the algorithm-manager. It is the component that is going to read and write data in the Kernel Platform so each time you make a read or a write using the `.format("telefonica")` in your algorithm, it will launch a data-io job. If you didn't persist your dataframe in Spark and needs to be computed again, it will read your data again from the EventHub.

- In write requests, it parses the data to ensure that it is syntactically valid and is well adjusted to the expected schema, otherwise the incorrect records are marked and stored as malformed.
- In read requests, it supports pushdown filters to avoid reading all of the rows/columns that exist in S3/WASB or Postgres. It will also filter any personal data that we do not have access based on GDPR restrictions. Invalid records that not match the schema can be requested to be dismissed or included into the read data, depending on client request.

There are some metrics that are exported to Prometheus:

- `spark_read_discards` : This is the number of rows that were discarded by the Kernel Platform when reading.
- `spark_write_discards` : This is the number of rows that were discarded by the Kernel Platform when writing.
- `spark_read_valid` : This is the number of rows that were valid when reading.
- `spark_filter_gdpr` : Number of rows after the filter GDPR.
- `spark_filter_partitions_to_write` : Number of rows after filtering data not in partitions to write.
- `spark_records_written` : Number of records written.

Processing environment services

> NOTE: this system will be deprecated in favor of Databricks for algorithms using the SDK v0.5+.

The local processing environment or, in short, the processing environment allows data processing in the Kernel Platform. Basically, it sets up the resources (i.e. VMs, etc) to launch a Spark cluster (drivers and controllers) and the EventHub used to read/write data to/from the Data-IO layer. When the processing finishes the resources are torn down. The processing environment is related to the services explained in this section and, additionally, it has a specific dependency with the Data-IO and Data Control layers and the following system services: graphite-exporter.

Consents Sync System

Entities

The consents sync system has been designed to keep users' consents in the Kernel Platform in sync with the OB's consents view.

Algorithms

Consents Sync Algorithm

This algorithm is in charge of checking the differences between consents in the Kernel Platform (stored in the `Consent_State` dataset) and OB's consents ingested in `Consent_State_Candidate`. The output of this algorithm is the generation of consent events as the result of the modification, addition or deletion of consents with the algorithm execution timestamp.

Consent State Exporter Algorithm

This algorithm is in charge of merging the new consent events from the `Consent_Event` dataset with the `Consent_State` dataset and keep the consents state history in the `Consent_State_History` dataset. Additionally if the OB is the master of consents, then the state of consents will be written to the `<core>.baikal.core.consents.personal` Kafka topic in order to keep the Kernel Platform consents database in sync.

Consent Exporter

This algorithm is in charge of reading the Kernel Platform's consents from the `<core>.baikal.core.consents.personal` Kafka topic and write consent events to the `Consent_Event` dataset.

> This algorithm only exists when the Kernel Platform is the master of consents.

Services

Privacy M

This service is subscribed to the end-of-ingestion notifications for datasets `Consent_Event` and `Consent_State_Candidate` . When the `Consent_State_Candidate` dataset is ingested this service will trigger the execution of the `consents-sync` algorithm. When the `Consent_Event` dataset is ingested , then the `consent-states-exporter` algorithm will be launched.

Both algorithm executions must be atomic. If an ingestion event is triggered while one of this algorithms is running this service must wait until the current algorithm ends before launching the next execution.

> You can find the mapping between the dataset ingestion and algorithm schedule information in the config map of this service.
>
> **Notifications API**

This service needs to read the changes of consents from the `<core>.baikal.core.consents.personal` Kafka topic in order to guarantee the GDPR between the user's events and the subscribed clients, by notifying these clients of all changes in consents.

Consent Event Manager

This service is in charge of notifying the subscribed clients of changes in user's consents.

> This notification will only be produced if the event was marked as notifiable.

Troubleshooting

What can I do if an algorithm fails?

If one of the consent algorithms fails it can be relaunched manually by checking the algorithm arguments in the `algorithm-manager` pod logs or by checking the corresponding `<core>.baikal.core.executions` Kafka topic message.

How can I prevent the notification of consents c

If the OB wants to ingest an initial load of consents without notifying the subscribed clients, the main class of the `consents-sync` algorithm must be changed to `com.telefonica.baikal.consentssync.MainInitialLoad` using the algorithm manager API. Remember that the main class must be restored to `com.telefonica.baikal.consentssync.ConsentsSync` when the initial load ends.

Audit System

The audit system is composed by 4 datasets: `Audit_Api`, `Audit_Sessions`, `Audit_Authorization`, `Audit_Data` and the algorithm `audit-exporter`.

Datasets

- **Audit_Data**: Stores all the activity related to the read/write access to the platform's data, including total number of bytes and records involved, the scope and purposes, execution resources and time, partitions read or written, client identifier …
- **Audit_Api**: Stores all the API requests invoked in the platform, with the scopes and purposes, request and response size, user and client identifier, HTTP endpoint and method …
- **Audit_Authorization**: Stores all the tokens generated int he platform, based on the grant types: jwt-bearer, authorization_code, refresh_token tokens are registered. The dataset also stores the scopes and purposes, the client and user identifier, the JTI …
- **Audit_Sessions**: Stores all the logins established with the IdP, with scopes and purposes, client and user identifier, the JTI…

Algorithm

The audit-exporter algorithm is executed hourly through the schedule provisioned in the platform. It is executed every hour at 50 minutes past the hour. This algorithm reads data from the Kafka Core and compute the audit datasets. The algorithm has an argument to define the mode/processors to run, the available options are: `sessions`, `authorizations`, `data`, `api-accesses`, `all` (default).

Mode/Processor	Read Kafka Topics	Generate Dataset
sessions	.baikal.core.events-authserver-idp-sessions	Audit_sessions
authorizations	.baikal.core.events-authserver-authorizations	Audit_Authorizations
api-accesses	.baikal.core.events-apigw	Audit_Api
data	.baikal.core.dataset-action, baikal.core.data-metrics, baikal.core.executions	Audit_Data

- Kafka Topics, source of data

  | Kafka Topic | Source of data | Description of data | |---|---|---| | .baikal.core.events-authserver-idp-sessions | authserver | Audit_sessions | | .baikal.core.events-authserver-authorizations | authserver | Audit_Authorizations | | .baikal.core.events-apigw | apigw | Audit_Api | | .baikal.core.executions | data-api, algorithm-manager | Algorithm execution info: time, status | | .baikal.core.data-metrics | data-api, data-io | Metrics about data read/writes: partitions, bytes, records... | | .baikal.core.dataset-action | data-api | Start/End of data read/writes in/from Kernel Platform |

The audit-exporter algorithm also read/write the state inside the dataset (Algorithm_State, v1), in this dataset is stored the last processed timestamp for each processor. The timestamp is used to read the specific time from Kafka.

Events Manager

consent-events-manager

The consent-events-manager synchronizes consents, purposes and pi-scopes with external systems.

It is subscribed to the internal changes of these entities using Kafka. Then, using the Notifications System, it sends an event notifiyng the change outside. In the opposite direction, this service is subscribed to external changes using the Notifications System. When an external system creates an event about changes in GDPR entities, the consents-events-manager will be notified and publish the necessary messages in Kafka in order to propagate the changes in the rest of the services.

user-events-manager

The user-events-manager translates user life cycle events to Kafka Events.

It is subscribed to some ob-events associated with user life cycle using the notifications system. When a notification arrives, it inserts an event in Kafka. Rest of services can react to these Kafka events.

Notifications System

Entities

The notification system is based on three types of instances: subscriptions, events, and notifications. The following is an example of the role of each type of entity:

The OBs (or any entity that has access to broadcast events) may issue events in the notification system in order to notify anyone who is interested in changes or specific events. For example, the OB's, when it cancels a customer, it can create user_removal events indicating the user_id of the user to delete. On the other hand, a user of the notification system will create subscriptions for a specific event type. For example, Aura could create a subscription for the user_removal event type, which indicates that an end customer has been deleted. Finally, when for a broadcast event there is a subscription that matches, a notification is generated and it will be sent to the http address configured by the subscription.

Some of the characteristics of each of the instances are explained in more detail below:

Events

Events will be sent to symbolize that something has happened in the system. Events always belong to a certain event_type and it can contain relevant information in the field called payload. An example of an event that symbolizes the deletion of a user is as follows:

```
{
    "id": "4865f00a-d146-4a23-acb9-4ca33ad8df40",
    "payload": {
        "reason": "It was a test user"
    },
    "creation_date": "2019-09-23T14: 41: 56Z",
    "user_id": "31d7a7e2-896d-11e7-bb31-be2e44b06b34"
}
```

The user_id field is an optional value and it is used to mark events with personal information. This value goes outside the payload for design reasons. In the swaggers that define each API about notifications, you can see the format of each field in the payload. Currently, we check some types such as dates or uuid to confirm that when an event is created, these formats are met. In case of creating an event with a wrong payload, the API will return error.

Subscriptions

In order to receive notifications of an event type (for example user_removal) we need to create a subscription. There are several operations to manage a subscription:

- Create subscription
- Remove subscription given a subscription_id
- Get a subscription given a subscription_id
- Get all subscriptions associated with an event_type

The subscription model has this format:

```
{
    "id": "31d7a7e2-896d-11e7-bb31-be2e44b06b34",
    "owner_id": "aura",
    "http_callback": "https://my-service.telefonica.com/subscriptions/callback",
    "event_type": "user_removal",
    "filters": [
        {
            "key": "user_id",
            "value": "31d7a7e2-896d-11e7-bb31-be2e44b06b34"
```

```
        }
    ]
}
```

This subscription would symbolize a subscription for all user_removal events that have a user_id field with the value 31d7a7e2-896d-11e7-bb31-be2e44b06b34. This subscription symbolize a subscription for all user_removal events that unsubscribe the user 31d7a7e2-896d-11e7-bb31-be2e44b06b34

The http_callback field is an optional field. In that field, you can define the uri where the notifications that match will be sent. On the other hand, in case the http_callback field is not defined, we are going to use it as a URI to send notifications to the http_callback field defined in the Application that was used to create the subscription. We can say then that there are two ways to define the http_callback

- I create a subscription without http_callback using an Application called Aura that defines an http_callback "https://my-service.telefonica.com/subscriptions/callback". The latter http_callback will be used. If the Application does not have http_callback defined, an error will be returned.
- I create a subscription with http_callback => this http_callback will be used to send notifications.

In addition to http_callback, in an Application you can also define a management_http_callback. This callback is not used to send notifications of events. It is used to send administration notifications. More information in the Management Notifications section.

Notifications

A notification is the result of the match between a subscription and an event. This instance will be sent to the configured http_callback. The notification contains the reference of the event with which the match was made and to subscription. A notification has the following form:

```
{
    "id": "31d7a7e2-896d-11e7-bb31-be2e44b06b34",
    "owner_id": "aura",
    "subscription_id": "31d7a7e2-896d-11e7-bb31-be2e44b06b34",
    "creation_date": "2019-10-23T14: 41: 56Z",
    "event": {
        "event_type": "user_removal",
        "user_id": "31d7a7e2-896d-11e7-bb31-be2e44b06b34",
        "payload": {
            "reason": "It was a test user"
        },
        "owner_id": "spain-ob",
        "operation_id": "7dcfa3ab-159f-41f5-ab69-1975f987ef4f",
        "id": "4865f00a-d146-4a23-acb9-4ca33ad8df40",
        "creation_date": "2019-09-23T14: 41: 56Z",
        "x-correlator": "65bd5bba-f607-408b-83f7-24e337e488ea"
    },
    "purposes": ["my-purpose"]
}
```

The purposes field will indicate, in the case of an event with personal information, the purposes to which you have access. For more information consult the GDPR filtering section

In addition to event notifications, there are also administration notifications. You can have more information in the Management Notifications section

Services

events-provisioner

This service is part of the Notifications System. It is a Kafka events topic consumer. When an event injection request is received by the Notifications API, it logs an event message at the events topic. The events-provisioner is responsible for matching all subscriptions according to the event message.

This service uses an in-memory subscription matching algorithm. All subscriptions are managed in a a multi-tiered (heap and disk) map. On application start, the component retrieves all the subscriptions from the subscriptions Kafka topic and loads them into the map, so it can take a while until the events-provisioner actually starts processing events.

The events-provisioner needs to be configured with the amount of expected subscriptions. If the actual number of subscriptions exceeds the configured one, the service will continue to work but with a degraded performance. The amount of memory needed will be automatically configured based on this setting. Please make sure your common nodes are big enough to fit the events-provisioner. If they aren't big enough, the pod will remain in the Pending state forever and the deployment will not succeed.

The number of expected subscriptions is configured in the Kernel Platform config, in the `service_configs['events-provisioner']['expected-subs']` YAML value. This value must be a non-zero, unsigned integer.

notifications-api

This service is part of the Notifications System. It exposes the Notifications API. It receives **events** and **subscriptions** and writes them to the associated Kafka topics. It also accepts queries on the events, subscriptions and **notifications** sent.

notifications-retrier

This service is part of the Notifications System. The Notifications System can fail for different reasons when it tries to send a notification (network issue, unavailable remote system, etc). This service is in charge of rescheduling the notifications to try to send them again.

It consumes events from the topic that contains the failed notifications, waits a few seconds (following an exponential backoff of `2^attempts` seconds) and injects the notification to the notifications topic to be rescheduled.

A notification is considered as failed when:

- The callback endpoint associated to the subscription returns an HTTP Status Code different to 2XX.
- The Kernel Platform doesn't receive an answer in 20 seconds.

See "Retries in the notifications system" and "How to check if the Kernel Platform sent a notification to a user" in the Kernel Platform Community Support center.

GDPR Filter

In order to comply with the GDPR law, the notifications generated are filtered. This functionality is activated for all events sent in version v3. The filtering of notifications by GDPR only applies to events with personal information, that is, events with the user_id informed.

Each type of event is protected by a list of pi-scopes. These pi-scopes can be seen in the swagger in the pi-scopes-related field at each event sending endpoint.

For each event with personal information that arrives, all the purposes associated with its pi-scopes-related are searched. With the purposes found, all subscriptions that matched are filtered. We only have the subscriptions that were created by Applications that contained at least some of the purposes. Finally, in case of not being automatic purposes, we are left only with the purposes for which there is consent. A notification will be generated for each of the filtered subscriptions. Each notification will have in the "purposes" field the list with the purposes for which it is accessed. If the purposes field does not appear, it means that the notification is not associated with an event with personal information.

Portal

portal-backend

The portal-backend has one unique responsibility: Login from users who use the developer portal (portal-ui).

To complete the login process, it needs to talk to the OpenID Connect (OIDC) provider configured in the global vault at deployment time. This is usually the corporate O365. Once the user is authenticated with OpenID, it talks to the authserver to obtain a token with the necessary permissions (scopes). The token is passed to the portal-ui to show the website.

portal-ui

The portal-ui is a simple stateless service that serves the Kernel Platform administrative user interface. It uses a nginx server to serve a single-page application. This application uses the portal-backend to get a token and accesses the Kernel Platform APIs just as any other API client would do.

Algorithms

Audit Exporter

For further information about this algorithm, please refer to this section.

Aura Dataset Importer

The aura-dataset-importer algorithm is executed every day at 5:15am through the schedule provisioned in the platform.

This algorithm reads Aura data in `CSV` format from the *aura conversations* storage account and computes the `Aura_Recognizer` and the `Aura_User` datasets. The algorithm has de following arguments:

Mode/Processor	Read Kafka Topics
--path_input	The directory where the Aura CSV data is stored (`/AURA_DATA//USER` or `/AURA_DATA//RECOGNIZER`)
--date	The date from which you want to obtain the data
--dataset	The execution mode, could be `recognizer` or `user` .

There are two schedules configured for this algorithm: `recognizer` and `user` data gathering.

> WARNING: The name of the Aura CSV files is really important. The algorithm is using this file naming in order to generate the `MODULE_CD` and `HOST_ID` datasets columns

Consents Sync & Consents Exporter & Consent States Exporter

For further information about these algorithms, please refer to this section.

Data Importer

The data-importer algorithm was designed for QA and performance testing purposes. This algorithm uses the spark SDK in both telefonica-external and telefonica spark data source modes. This algorithm can read data form file system in `AVRO` or `CSV` format and ingest it to the platform, or read data from the platform and write it into the file system.

Data IO

For further information about this algorithm, please refer to this section.

Dataset Tools

The dataset-tools algorithm is an utility that allow to perform dataset maintenance operations (retention, time-travel, etc)..

KPIs

For further information about these algorithms, please refer to this section.

Malformed Dataset Migrator

The malformed-dataset-migrator algorithm is an utility for migrating batch's datasets that are stored applying logical-types to malformed data compliance.

> WARNING: this algorithm is designed to be running in pre delta lake scenario and must not be running after Kivu release.

Metrics exporter

For further information about this algorithm, please refer to this section.

Thor

For further information about this algorithm, please refer to this section.

Thor: Data Quality

The Kernel Platform provides mechanisms to evaluate and generate metrics on data quality of datasets. The data analysis is performed by the Thor algorithm. This algorithm reads the datasets and gets their metadata to compute dataset quality metrics that are stored in the dataset `thor_results` version 2.

Validations

Different validations can be computed. Some of them (the general ones) are applicable for all datasets, while others are executed only if the associated metadata has been declared in the dataset schema. Moreover, validations can be declared dynamically for a specific dataset. All of them are explained in the following sections.

General

Some general validations and statistics are computed for all of the dataset fields:

- **NULL counts**: Counts the number of NULL rows in that column.
- **NOT_INFORMED counts**: Counts the number of NOT_INFORMED rows in that column.
- **MAX/MIN** (number, date and timestamp data type): Max and min value for that column.
- **AVERAGE** (number data type): Average value for that column.

These validations always compute their results when a dataset is analyzed.

Joins

The joins validations compute how many records have values that do not match between two datasets that have been declared to have matching values through the Avro4P.

DatasetA

ColumnA	ColumnB
abcd	11
azx	22
alx	al

DatasetB

ColumnZ	ColumnY
abcd	ab
azx	az
alx	al

- A join of DatasetA(columnA) with DatasetB(columnZ), returns no-match count of 2 values (abcd, azx).

This metric should have a value equal to zero, for datasets with all records honoring the join declaration.

Not Sync In Time

The not sync in time validation counts the number of records that have different dates in their time fields according to dataset frequency. For example:

date	timestamp	ColumnY
2020-01-01	2020-01-01 01:00:00	abz
2020-01-02	2020-01-02 01:00:00	abx
2020-01-03	2020-01-02 01:00:00	aby

This dataset has DAY frequency configured. The not-sync-in-time validation checks all the dataset fields marked with `x-fp-time-dimension` and, for each record, counts how many of them are not in sync. In this example, the last record has a timestamp corresponding to a different date of that informed in the date field, so this would increase the overall count by one.

This metric should have a value equal to zero, for datasets with all records having in synch time fields.

Dynamic

The dynamic validations allow to define custom validations using SparkSQL. Thor creates a temporary spark view with `validation_data_table` name that contains all filtered data from the target dataset. Also, validations allow to add more dataset sources to build more complex queries.

Example: Distribution of customers by segments.

- `validation_data_table` : Contains the data of Customer dataset with the applied filters.
- `D_Segment_5` source: Contains all the data from D_Segment_5.

```
{
  "id": "CUSTOMER_ID-distribution_by_SEGMENT_ID",
  "description": "Distribution(json{SEGMENT: text value, COUNT_CUSTOMERS: Int}) of number of CUSTOMER_ID by SEG
MENT_ID",
  "dataset_id": "Customer",
  "dataset_version": 5,
  "sources": [
    {
      "name": "D_Segment_5",
      "dataset_id": "D_Segment",
      "dataset_version": 5
    }
  ],
  "field": "CUSTOMER_ID",
  "sql": "SELECT
            to_json(collect_list((LOCALSEGMENT_GLOBALSEGMENT,COUNT_CUSTOMERS))) as distribution
          FROM
            (SELECT
              COUNT(*) AS COUNT_CUSTOMERS,
              CONCAT(A.SEGMENT_ID,'_', DIM.GBL_SEGMENT_ID) AS LOCALSEGMENT_GLOBALSEGMENT
            FROM validation_data_table A
            LEFT JOIN D_Segment_5 DIM
              ON A.SEGMENT_ID=DIM.SEGMENT_ID
            GROUP BY 2)",
  "metrics": ["distribution"]
}
```

The dynamic validations are provisioned using the dataset validation endpoints.

Unique

The unique validation counts repeated tuples of fields. For example:

DatasetA

ColumnA	ColumnB
abcd	11
azx	22
art	NULL
art	NULL
abcd	art
abcd	11

- UNIQUE(ColumnA): 3 (abcd is repeated twice + art once)
- UNIQUE(ColumnA, ColumnB): 2 (abcd,11 and art,NULL are both repeated once)

The unique validation checks all the dataset fields marked with `x-fp-unique-constraints` in their schema.

This metric should have a value equal to zero, for datasets with all records respecting the uniqueness rules defined in its schema.

Execution

At the moment, the Thor algorithm is executed manually. Execution needs a dataset list with datasetID and version splited by "-" to identify the data to analyze and an optional filters start date and end date to analyze a specific interval of the dataset. If you only want to validate the dataset for one date you have to indicate in start date.

Thor execution args

Property	Description	Required	Example
dataset-list	List of Dataset ID and dataset major version	x	"Customer-5,Mobile_Line-6"
check-list	List of metrics to check		"General,Join,Dynamic,Unique,Time"
filter-startdate	Filter start date using in ISO8601 format		2020-01-01T00:00:00Z
filter-enddate	Filter end date using in ISO8601 format		2020-01-15T00:00:00Z

- Algorithm Execution Endpoint

Results

The data quality results are stored in a batch dataset `thor_results`, version 2. The Avro 4P schema of this result dataset is:

```
{
  "doc": "A basic schema for representing Thor results",
  "fields": [
    { "name": "TIMESTAMP", "type": { "type": "string", "logicalType": "datetime" } },
    { "name": "DATASET_ID", "type": "string" },
    { "name": "DATASET_VERSION", "type": "string" },
    { "name": "FIELDS", "type": ["null", { "items": "string", "type": "array" }] },
    { "name": "FILTER_DATE", "type": ["null", { "type": "string", "logicalType": "datetime" }] },
    { "name": "METRIC_NAME", "type": "string" },
    { "name": "METRIC_VALUE", "type": "string" }
  ],
```

```
  "name": "Thor_results",
  "namespace": "com.telefonica",
  "type": "record",
  "x-fp-version": "2.0.0"
}
```

The dataset is partitioned by: `DATASET_ID`, `DATASET_VERSION`, `FILTER_DATE`. Results can be extracted by reading the dataset.

Metrics

Metric	Scope	Description	Exa
`total_count`	GLOBAL	Count of rows.	"
`null_total_count`	FIELD	Count of NULL values.	'
`null_percentage`	FIELD	% of NULL values.	"25
`not_informed_total_count`	FIELD	Count of NOT_INFORMED values.	"
`not_informed_percentage`	FIELD	% of NOT_INFORMED values.	"5(
`highest_value`	FIELD	The max value in this field (number and dates).	"2("202 01T2(
`smallest_value`	FIELD	The min value in this field (number and dates).	"1", (01T2(
`average_value`	FIELD	The average value in this field (number).	"10
`unique_constraint_repeat_count`	FIELD	The number of repeated values.	'
`unique_constraint_repeat_percentage`	FIELD	% of repeated values.	"25
`time_not_sync_count`	FIELD	Count of rows with out-of-sync times.	"
`time_not_sync_percentage`	FIELD	% of rows with out-of-sync times.	"5(
`not_cross_with_${DATASET_ID}_${DATASET_VERSION}_total`	FIELD	Count of rows that don't match with other dataset.	"
`not_cross_with_${DATASET_ID}_${DATASET_VERSION}_percentage`	FIELD	% of rows that don't match with other dataset.	"5(
`not_cross_with_${DATASET_ID}_${DATASET_VERSION}_examples`	FIELD	Examples of values that don't match with other dataset	"(MS -> (MSI(>
`dynamic_${VALIDATION_ID}_${VALIDATION_METRIC}`	GLOBAL/FIELD	Value of the dynamic validation metric.	

`total_malformed_count`	GLOBAL	Count of rows with any malformed field.	"
`total_malformed_percentage`	GLOBAL	% of rows with any malformed field.	"5
`malformed_count`	FIELD	Count of malformed values.	"
`malformed_percentage`	FIELD	% of malformed values.	"3

Monitoring

Monitoring systems allow to check not only if an application is alive and healthy but also to know how a service is working internally. This guide is intended to provide useful information in order to ensure an efficient Kernel Platform monitoring.

The Kernel Platform offers the following tools to help monitoring the status of the platform:

- Different metrics and real-time alerts defined on top of the metrics.
- Dashboards to access the metrics in a visual way.
- Alerts summary
- Aggregated logs.
- Tracing support.
- Spark history in order to see the algorithm executions DAGs.

Integration with third-party services and monitoring tools is possible using the Kernel Platform public APIs. It is important to use only the officially exposed APIs because ad-hoc integrations could break without notice.

Endpoints summary of monitoring tools

- Grafana: https://backoffice.$CORE.baikalplatform.com/dashboard
- Kibana: https://backoffice.$CORE.baikalplatform.com/logs
- Jaeger: https://backoffice.$CORE.baikalplatform.com/tracing
- Pyroscope: https://backoffice.$CORE.baikalplatform.com/profiling

Metrics

Prometheus gathers the Kernel Platform metrics and decides when to send an alert. Metrics include not only information specific to the Kernel Platform services (e.g. number of HTTP requests or average latency), but also data about the infrastructure that sustains the Kernel Platform (e.g. CPU and memory usage) or any other indicator.

As other metric systems, Prometheus stores all its data in a time series database. However, it offers a multi-dimensional data-model and a powerful query language (PromQL) allowing system administrators to generate more accurate reports. Prometheus makes its metrics available to be queried using its own API.

ℹ️ Prometheus is the only official public interface offered to integrate the Kernel Platform metrics with third-party systems, if needed.

The official Prometheus web UI is available at https://metrics.$TENANT.baikalplatform.com and includes not only an option to run queries, but also information about the service such as the targets it is scraping, etc.

Prometheus is tightly integrated with other Kernel Platform services:

- **Grafana**, one of the clients that uses Prometheus to visualize the metrics in dashboards.
- **Alertmanager**, a service in charge of sending email alert notifications.

ℹ️ You can query the Prometheus API to get the list of all available metrics at the following endpoint, authenticating the request with one of the backoffice users defined in the deployment config file (`service_configs.backoffice.users`) used during the Kernel Platform deployment:

> https://metrics.$TENANT.baikalplatform.com/api/v1/label/__name__/values

Internal metrics usage

Since Prometheus exposes the official API to consume metrics, many services in the Kernel Platform also use it for different purposes.

KPI reports are periodically generated with the help of two services, prometheus-exporter and prometheus-kafka-adapter, that take care of gathering the metrics that can be useful for purposes other than monitoring and leave them in a Kafka topic so they end up persisted in a long-term storage such as the cloud object storage.

Algorithms in the processing environments also consume the Prometheus API to return the status of a particular data ingestion process.

Metrics will have additional internal usages in the future, such as helping the platform to scale up and down using custom metrics to determine the platform's health.

Alerts

As previously stated, Prometheus has a list of alert rules that are part of the platform configuration. Alerting rules allow you to define alert conditions based on Prometheus expression language. It is possible to edit the Kernel Platform alert rules but changes are lost in a redeployment for now. If you think an alert is important and should be part of the platform, let us know so we can officially include it.

Alerts are sent via email, using a global SMTP server managed by the Kernel Platform Team.

Alerts are disabled (silenced) during Kernel Platform deployments to avoid false positives due to services that need to be restarted, etc.

External Forwarding To automate the sending of alerts to external remote servers, we need add a new section "alertmanager" in the configuration service. Example: service_configs: alertmanager: external_forwarding: remote_servers:

```
                - name: "unique-identifying-name-0"
                  hostname:
                  port:
                - name: "unique-identifying-name-1"
                  hostname:
                  port:
                  tls_config:
                  ca_certificate: |-
                  certificate: |-
                  private_key: |-
```

2 types of targets (remote servers) can be declared.

- secure_tls with normal certificates (no require tls_config).
- insecure_tls with self-signed certificates (require tls_config).

Required fields are:

- name: Type String, required.
- hostname: Type String, required.
- port: Type Int, between 1 and 65535, required. Optional fields, only necessary to tls_insecure (self-signed certificates)
- tls_config: label to initialize tls_insecure mode.
- ca_certificate: Type String.
- certificate: Type String.
- private_key: Type String.

Dashboards

Grafana is an opensource tool integrated in the Kernel Platform to visualize the available metrics.

The Kernel Platform comes with many pre-installed dashboards. They are labeled as "baikal" and they shouldn't be modified nor removed because they are overwritten on each new installation.

Some dashboards show information regarding Kernel Platform Services and others offer low-level information about the Kubernetes cluster and the cloud infrastructure. If you think some info is missing, feel free to contact the Kernel Platform Team to evaluate whether it could be officially included as part of the platform.

The official Grafana web UI is available at:

> https://backoffice.$CORE.baikalplatform.com/dashboard

Alerts summary

algorithm_environment_deletion_failed

- severity: critical
- origin: core
- summary: ALGORITHM ENVIRONMENT DELETION WARNING
- description: Execution environment for algorithm `$labels.client_id` couldn't be deleted.

algorithm_failed

- severity: warning
- origin: core
- summary: ALGORITHM FAILING
- description: The `$labels.client_id` algorithm has failed.

api_endpoint_4xx_error_anomaly_detection

- severity: warning
- origin: core
- summary: Requests for job `$labels.job` with code: `$labels.code` are outside of expected operating parameters.
- description: Requests for job `$labels.job` in service: `$labels.service` with code: `$labels.code` about tenant: `$labels.tenant` are outside of expected operating parameters.

api_endpoint_5xx_error_anomaly_detection

- severity: warning
- origin: core
- summary: Requests for job `$labels.job` with code: `$labels.code` are outside of expected operating parameters.
- description: Requests for job `$labels.job` in service: `$labels.service` with code: `$labels.code` about tenant: `$labels.tenant` are outside of expected operating parameters.

api_endpoint_latency_anomaly_detection

- severity: warning
- origin: core
- summary: Requests for job `$labels.job` in service: `$labels.service` are outside of expected operating parameters.
- description: Requests for job `$labels.job` in service: `$labels.service` tenant: `$labels.tenant` are outside of expected operating parameters.

api_rate_limit_reached

- severity: warning
- origin: core
- summary: API rate limit reached

- description: API rate limit reached on path `$labels.path`

apigw_audit_info_not_emitting

- severity: critical
- origin: core
- summary: APIGW NOT EMITTING AUDIT INFO
- description: Apigw is not emitting audit information in `settings.services.kafka.topics['events-apigw'].name` topic

apigw_certificate_expires_alert_critical

- severity: critical
- origin: core
- summary: APIGW CERTIFICATE IS ABOUT EXPIRE WITHIN 7 DAYS
- description: Apigw `$labels.entity` with `$labels.id` id for `$labels.service` service is about expire within 7 days

apigw_certificate_expires_alert_warning

- severity: warning
- origin: core
- summary: APIGW CERTIFICATE IS ABOUT EXPIRE WITHIN 30 DAYS
- description: Apigw `$labels.entity` with `$labels.id` id for `$labels.service` service is about expire within 30 days

apigw_nginx_worker_processes_restarting

- severity: critical
- origin: core
- summary: APIGW RESTARTING WORKER PROCESSES
- description: Apigw is restarting worker processes abnormally.

apigw_some_nginx_worker_processes_restarting

- severity: warning
- origin: core
- summary: APIGW SOME RESTARTING WORKER PROCESSES
- description: Some Apigw worker processes are being restarted.

audit_exporter_algorithm_failed

- severity: critical
- origin: core
- summary: AUDIT-EXPORTER ALGORITHM FAILING
- description: The `$labels.client_id` algorithm has failed.

authserver_99_latency_req

- severity: warning
- origin: core
- summary: AUTHSERVER - HIGH LATENCY (>2.5s)
- description: The authserver is suffering abnormal latencies (99%) with a value of `$value`

authserver_authserver_authentication_code_failure_total_anomaly_detection

- severity: warning
- origin: core
- summary: Requests for job `$labels.job` in `$labels.client_id` are outside of expected operating parameters.
- description: Requests for job `$labels.job` in `$labels.client_id` client_id about tenant: `$labels.tenant` are outside of expected operating parameters.

authserver_authserver_idp_success_total_anomaly_detection

- severity: warning
- origin: core
- summary: Requests for job `$labels.job` in `$labels.idp_broker` are outside of expected operating parameters.
- description: Requests for job `$labels.job` in `$labels.idp_broker` about tenant: `$labels.tenant` are outside of expected operating parameters.

authserver_fail_percent_req

- severity: critical
- origin: core
- summary: AUTHSERVER - HIGH ERROR RATE (>25%)
- description: Authserver endpoint `$labels.uri` is returning an high number of error responses for `$value` % of requests in `$labels.tenant` with client: `$labels.client`

authserver_idp_wellknown_failed

- severity: critical
- origin: core
- summary: AUTHSERVER - IDP DISCOVERY WELL-KNWON ENDPOINT ERROR
- description: The authserver is receiving too many errors from the discovery well-known endpoint of IDPs

available_nodes

- severity: critical
- origin: system
- summary: AVAILABLE KUBERNETES NODES UNDER 60%
- description: The number of available nodes is `$value` %. Some nodes could be in unready state.

azure_event_hub_quota_exceeded_errors

- severity: warning
- origin: system
- summary: Azure EventHub `$labels.instance_name` has quota exceeded errors in `$labels.entityname` namespace
- description: Azure EventHub `$labels.instance_name` has quota exceeded errors in `$labels.entityname` namespace.

azure_event_hub_server_errors

- severity: warning
- origin: system
- summary: Azure EventHub `$labels.instance_name` has server errors in `$labels.entityname` namespace
- description: Azure EventHub `$labels.instance_name` has server errors in `$labels.entityname` namespace.

azure_event_hub_throttled_requests

- severity: warning
- origin: system
- summary: Azure EventHub `$labels.instance_name` has throttled requests in `$labels.entityname` namespace
- description: Azure EventHub `$labels.instance_name` has throttled requests in `$labels.entityname` namespace.

azure_high_snat_ports_used_on_load_balancer

- severity: warning
- origin: system
- summary: AZURE LOAD BALANCER HIGH SNAT PORTS USED ON
- description: High SNAT ports used on `$labels.backendipaddress` - `$labels.protocoltype` - `$labels.frontendipaddress` in `$labels.instance_name` load balancer from `$labels.resource_group` resourge group. Approaching SNAT port exhaustion.

azure_load_balancer_snat_connections_failures

- severity: critical
- origin: system
- summary: AZURE LOAD BALANCER SNAT CONNECTION FAILURES
- description: SNAT Connection failures on `$labels.instance_name` load balancer from `$labels.resource_group` resourge group. AKS nodes are failing to initiate outbound connections. Has been reached the connection limit or SNAT port exhaustion.

azure_postgresql_cpu_percent_warning

- severity: warning
- origin: system
- summary: Azure PostgresSQL `$labels.instance_name` high CPU use WARNING
- description: Postgres instance `$labels.instance_name` has an high CPU use: (`$value` %) in the last 10 minutes

azure_postgresql_io_consumption

- severity: critical

- origin: system
- summary: Azure PostgresSQL `$labels.instance_name` high IO Consumption use CRITICAL
- description: Postgres instance `$labels.instance_name` has an high IO Consumption use: (`$value` %) in the last 5 minutes

azure_postgresql_memory_percent

- severity: critical
- origin: system
- summary: Azure PostgresSQL `$labels.instance_name` high MEMORY use CRITICAL
- description: Postgres instance `$labels.instance_name` has an high MEMORY use: (`$value` %) in the last 10 minutes

azure_postgresql_storage_percent_critical

- severity: critical
- origin: system
- summary: Azure PostgresSQL `$labels.instance_name` high storage use CRITICAL
- description: Postgres instance `$labels.instance_name` has an high STORAGE use: (`$value` %) in the last 10 minutes

azure_postgresql_storage_percent_warning

- severity: warning
- origin: system
- summary: Azure PostgresSQL `$labels.instance_name` high storage use WARNING
- description: Postgres instance `$labels.instance_name` has an high STORAGE use: (`$value` %) in the last 10 minutes

capacity_events_processed_total_critical

- severity: critical
- origin: core
- summary: EVENTS_PROCESSED IS CAPACITY 90%
- description: The events-provisioner is at 90% capacity to tenant: `$labels.tenant`

capacity_events_processed_total_warning

- severity: warning
- origin: core
- summary: EVENTS_PROCESSED IS CAPACITY 70%
- description: The events-provisioner is at 70% capacity to tenant: `$labels.tenant`

capacity_notifications_created_total_critical

- severity: critical
- origin: core
- summary: NOTIFICATIONS_CREATED IS AT CAPACITY 90%
- description: The notifications-api is at 90% capacity to tenant: `$labels.tenant`

capacity_notifications_created_total_warning

- severity: warning
- origin: core
- summary: NOTIFICATIONS_CREATED IS AT CAPACITY 70%
- description: The notifications-api is at 70% capacity to tenant: `$labels.tenant`

capacity_notifications_processed_total_critical

- severity: critical
- origin: core
- summary: NOTIFICATIONS_PROCESSED IS AT CAPACITY 90%
- description: The notifications-api is at 90% capacity to tenant: `$labels.tenant`

capacity_notifications_processed_total_warning

- severity: warning
- origin: core
- summary: NOTIFICATIONS_PROCESSED IS AT CAPACITY 70%
- description: The notifications-api is at 70% capacity to tenant: `$labels.tenant`

capacity_notifications_sent_total_critical

- severity: critical
- origin: core
- summary: NOTIFICATIONS_SENT IS AT CAPACITY 90%
- description: The notifications-api is at 90% capacity to tenant: `$labels.tenant`

capacity_notifications_sent_total_warning

- severity: warning
- origin: core
- summary: NOTIFICATIONS_SENT IS AT CAPACITY 70%
- description: The notifications-api is at 70% capacity to tenant: `$labels.tenant`

certificate_expires_alert_critical

- severity: critical
- origin: system
- summary: CERTIFICATE IS ABOUT EXPIRE
- description: Certificate for " `$labels.key_name` " is about expire within 7 days in Kubernets secret " `$labels.secret_namespace` / `$labels.secret_name` "

certificate_expires_alert_warning

- severity: warning

- origin: system
- summary: CERTIFICATE IS ABOUT EXPIRE
- description: Certificate for " `$labels.key_name` " is about expire within 30 days in Kubernets secret " `$labels.secret_namespace` / `$labels.secret_name` "

consent_algorithm_failed

- severity: critical
- origin: core
- summary: CONSENT ALGORITHM FAILING
- description: The `$labels.client_id` algorithm has failed.

container_image_pull_error

- severity: critical
- origin: system
- summary: PULLING IMAGE ERROR IN CONTAINER `$labels.container`
- description: Container `$labels.container` is having problems pulling images from its repository. It is not possible to run the service

container_memory_usage_near_limit

- severity: warning
- origin: system
- summary: CONTAINER MEMORY USAGE NEAR LIMIT
- description: A high memory volume near to the container resource limit is used by `$labels.pod` pool. Currently `$value` bytes used by the container.

coredns_down

- severity: critical
- origin: system
- summary: COREDNS DOWN CRITICAL
- description: Coredns down

coredns_errors_high

- severity: critical
- origin: system
- summary: COREDNS IS RETURNING SERVFAIL
- description: Coredns is returning SERVFAIL for `$value` of request

coredns_request_latency

- severity: critical
- origin: system
- summary: COREDNS HIGH LATENCY

- description: Coredns is returning high latency `$value` in request

dataset_tools_algorithm_failed

- severity: critical
- origin: core
- summary: DATASET TOOLS ALGORITHM FAILING
- description: The `$labels.client_id` algorithm has failed.

ddos_under_attack

- severity: critical
- origin: system
- summary: DDOS ATTACK DETECTED
- description: 4th Platform is under a DDoS attack.

filebeat_not_reading_files

- severity: critical
- origin: core
- summary: FILEBEAT NOT READING EVENTS
- description: Filebeat is not reading audit event files in its pod `$labels.kubernetes_pod_name` . Audit info might be lost.

filebeat_not_sending_to_kafka

- severity: warning
- origin: core
- summary: FILEBEAT NOT INJECTING EVENTS
- description: Filebeat is not injecting audit events in kafka

fluentbit_error_emitting

- severity: critical
- origin: system
- summary: FLUENTBIT EMITTING WITH ERRORS
- description: Fluent-bit pod `$labels.kubernetes_pod_name` is emitting with errors

fluentbit_not_emitting

- severity: critical
- origin: system
- summary: FLUENTBIT NOT EMITTING
- description: Fluent-bit pod `$labels.kubernetes_pod_name` is not emitting new logs

fluentbit_not_reading

- severity: critical
- origin: system
- summary: FLUENTBIT NOT READING
- description: Fluent-bit pod `$labels.kubernetes_pod_name` is not reading new logs

fluentd_not_emitting

- severity: critical
- origin: system
- summary: FLUENTD NOT EMITTING
- description: Fluentd node `$labels.host` is not emitting new logs

frequent_container_killed_by_memory_reason

- severity: warning
- origin: system
- summary: KUBERNETES FREQUENT CONTAINER RESTARTS (OOM) WARNING
- description: Container `$labels.container` on pod `$labels.pod` has been restarted `$value` times within the last hour.

frequent_container_restarts

- severity: warning
- origin: system
- summary: KUBERNETES FREQUENT CONTAINER RESTARTS WARNING
- description: Container `$labels.container` on pod `$labels.pod` has been restarted `$value` times within the last hour.

high_cpu_usage_on_hosts_critical

- severity: critical
- origin: system
- summary: HIGH CPU USAGE CRITICAL ON `$labels.kubernetes_io_hostname`
- description: `$labels.kubernetes_io_hostname` is using a LOT of CPU. CPU usage is `humanize $value` %.

high_cpu_usage_on_hosts_warning

- severity: warning
- origin: system
- summary: HIGH CPU USAGE WARNING ON `$labels.kubernetes_io_hostname`
- description: `$labels.kubernetes_io_hostname` is using a LOT of CPU. CPU usage is `humanize $value` %.

high_disk_io_pressure_on_hosts

- severity: critical
- origin: system
- summary: HIGH DISK IO PRESSURE WARNING ON `$labels.kubernetes_io_hostname`
- description: `$labels.kubernetes_io_hostname` has a high Disk IO pressure. Current Disk IO pressure is `humanize $value` %.

high_fs_usage_on_hosts_critical

- severity: critical
- origin: system
- summary: HIGH FILESYSTEM USAGE CRITICAL ON `$labels.kubernetes_io_hostname`
- description: `$labels.kubernetes_io_hostname` is using a LOT of FileSystem space in device `$labels.device` . FileSystem usage is `humanize $value` %.

high_fs_usage_on_hosts_warning

- severity: warning
- origin: system
- summary: HIGH FILESYSTEM USAGE WARNING ON `$labels.kubernetes_io_hostname`
- description: `$labels.kubernetes_io_hostname` is using a LOT of FileSystem space in device `$labels.device` . FileSystem usage is `humanize $value` %.

high_memory_usage_on_hosts

- severity: critical
- origin: system
- summary: HIGH MEMORY USAGE WARNING ON `$labels.kubernetes_io_hostname`
- description: `$labels.kubernetes_io_hostname` is using a LOT of Memory. Memory usage is `humanize $value` %.

high_persistent_volume_inode_usage

- severity: critical
- origin: system
- summary: HIGH PERSISTENT VOLUME INODE USAGE WARNING ON ' `$labels.kubernetes_io_hostname` ' by ' `$labels.persistentvolumeclaim`
- description: `$labels.persistentvolumeclaim` on `$labels.kubernetes_io_hostname` is using a LOT of persistent volume inodes. Persistent volume inode usage is `humanize $value` %.

high_persistent_volume_usage

- severity: critical
- origin: system
- summary: HIGH PERSISTENT VOLUME USAGE WARNING ON ' `$labels.kubernetes_io_hostname` ' by ' `$labels.persistentvolumeclaim`
- description: `$labels.persistentvolumeclaim` on `$labels.kubernetes_io_hostname` is using a LOT of persistent volume space. Persistent volume usage is `humanize $value` %.

jvm_thread_peak_critical

- severity: critical
- origin: system
- summary: JVM THREAD PEAK CRITICAL
- description: The number of threads used by application `$labels.app` in its pod `$labels.kubernetes_pod_name` is

compromising node stability. Stop this pod or node will be missed.

jvm_thread_peak_high

- severity: warning
- origin: system
- summary: JVM THREAD PEAK HIGH
- description: The application `$labels.app` is using an abnormal number of threads in its pod `$labels.kubernetes_pod_name`. Check this service it could compromise node stability.

keda_scaled_object_errors

- severity: critical
- origin: system
- summary: KEDA SCALED OBJECT ERRORS
- description: The `$labels.scaledObject` scaledObject in the `$labels.namespace` namespace is not scaling because errors have been detected.

keda_scaler_errors

- severity: critical
- origin: system
- summary: KEDA SCALER ERRORS
- description: The `$labels.scaledObject` scaledObject in the `$labels.namespace` namespace is not scaling because errors have been detected in the `$labels.scaler` scaler.

kpis_algorithm_failed

- severity: critical
- origin: core
- summary: KPIS ALGORITHM FAILING
- description: The `$labels.client_id` algorithm has failed.

kubelet_pleg_duration_high

- severity: warning
- origin: system
- summary: KUBELET PLEG TOO LONG TO RELIST
- description: The Kubelet Pod Lifecycle Event Generator has a 99th percentile duration of `$value` seconds on node `$labels.node`.

kubelet_pod_startup_latency_high

- severity: warning
- origin: system
- summary: KUBELET POD STARTUP LATENCY TOO HIGH
- description: Kubelet Pod startup 99th percentile latency is `$value` seconds on node `$labels.instance`.

kubernetes_apiserver_down

- severity: critical
- origin: system
- summary: NO KUBERNETES API SERVERS ARE REACHEABLE
- description: No API servers are reachable or all have disappeared from service discovery

kubernetes_apiserver_error_high_critical

- severity: critical
- origin: system
- summary: KUBERNETES API SERVER ON `$labels.instance` REQUEST ERROR CRITICAL
- description: Kubernetes API server on `$labels.instance` returns errors for `$value` % of requests

kubernetes_apiserver_error_high_warning

- severity: warning
- origin: system
- summary: KUBERNETES API SERVER ON `$labels.instance` REQUEST ERROR WARNING
- description: Kubernetes API server on `$labels.instance` returns errors for `$value` % of requests

kubernetes_apiserver_latency_high_critical

- severity: critical
- origin: system
- summary: KUBERNETES API SERVER ON `$labels.instance` REQUEST HIGH LATENCY CRITICAL
- description: The Kubernetes API server has a 99th percentile latency of `$value` seconds for `$labels.verb` (`$labels.resource`) on `$labels.instance`

kubernetes_apiserver_latency_high_warning

- severity: warning
- origin: system
- summary: KUBERNETES API SERVER ON `$labels.instance` REQUEST HIGH LATENCY WARNING
- description: The Kubernetes API server has a 99th percentile latency of `$value` seconds for `$labels.verb` (`$labels.resource`) on `$labels.instance`

log_oom_error_warning

- severity: warning
- origin: system
- summary: OutOfMemoryError Detect in containers: `$labels.containers`
- description: Detect `$value` OutOfMemoryError in containers `$labels.containers`

nginx_ingress_upstream_server_errors_responses_anomaly_detection

- severity: warning
- origin: system
- summary: ANOMALY SERVER ERRORS RESPONSES BY NGINX INGRESS DETECTED
- description: Anomaly server errors responses detected by Nginx Ingress for `$labels.upstream` upstream. Current ratio is outside of the expected operating parameters.

nginx_ingress_upstream_server_requests_anomaly_detection

- severity: warning
- origin: system
- summary: ANOMALY REQUEST PER SECONDS BY NGINX INGRESS DETECTED
- description: Anomaly requests per seconds detected by Nginx Ingress for `$labels.upstream` upstream. Current ratio is outside of the expected operating parameters.

nginx_ingress_waf_percentage_blocked_requests_anomaly_detection

- severity: critical
- origin: system
- summary: ANOMALY PERCENTAGE BLOCKED REQUEST BY NGINX INGRESS WAF DETECTED
- description: Anomaly percentage of blocked requests by `$labels.violation` Nginx Ingress WAF Policy detected. Current ratio is outside of the expected operating parameters.

node_not_ready_long_time

- severity: warning
- origin: system
- summary: LONG TIME NODE NOT READY STATUS
- description: Node `$labels.node` is in Not Ready status for a long time, probably due to kubelet outage.

portal_backend_credentials_expire

- severity: critical
- origin: core
- summary: PORTAL-BACKEND CREDENTIALS EXPIRE!
- description: Portal-backend, the provided client secret keys for app are expired

postgres_deadlocks_conflicts

- severity: critical
- origin: core
- summary: POSTGRES HIGH DEADLOCKS / CONFLICTS RATE
- description: Postgres instance `$labels.kubernetes_name` has an high number of `$labels.event` (`$value`) in the last 10 minutes

postgres_up

- severity: critical
- origin: core
- summary: POSTGRES NOT WORKING
- description: Postgres `$labels.kubernetes_name` has stopped working.

probe_down

- severity: critical
- origin: system
- summary: PROBE FAILING
- description: The endpoint `$labels.instance` is down or not reachable. The blackbox exporter could not validate `$labels.app` 's health.

prometheus_indexing_backlog

- severity: warning
- origin: system
- summary: PROMETHEUS INDEXING BACKLOG WARNING
- description: Prometheus is backlogging on the indexing queue. Queue is currently `$value | printf` %.0f``% full.

prometheus_not_ingesting_samples

- severity: critical
- origin: system
- summary: PROMETHEUS NOT INGESTING SAMPLES WARNING
- description: Prometheus has not ingested any samples in the last 10 minutes.

prometheus_notifications_backlog

- severity: warning
- origin: system
- summary: PROMETHEUS NOTIFICATIONS BACKLOG WARNING
- description: Prometheus is backlogging on the notifications queue. The queue has not been empty for 10 minutes. Current queue length: `$value` .

prometheus_persist_errors

- severity: critical
- origin: system
- summary: PROMETHEUS PERSIST ERRORS WARNING
- description: Prometheus has encountered `$value` persist errors per second in the last 10 minutes.

prometheus_persistence_pressure_too_high_24h

- severity: critical
- origin: system
- summary: PROMETHEUS PERSISTENCE PRESSURE 24H WARNING

- description: Prometheus is approaching critical persistence pressure. Throttled ingestion expected within the next 24h.

prometheus_persistence_pressure_too_high_2h

- severity: warning
- origin: system
- summary: PROMETHEUS PERSISTENCE PRESSURE 24H WARNING
- description: Prometheus is approaching critical persistence pressure. Throttled ingestion expected within the next 2h.

prometheus_rule_evaluation_slow

- severity: warning
- origin: system
- summary: PROMETHEUS RULE EVALUATION SLOW WARNING
- description: Prometheus has a 90th percentile latency of `$value` s completing rule evaluation cycles.

prometheus_series_maintenance_stalled

- severity: critical
- origin: system
- summary: PROMETHEUS SERIES MAINTENANCE WARNING
- description: Prometheus is maintaining memory time series so slowly that it will take `$value | printf` %.0f``h to complete a full cycle. This will lead to persistence falling behind.

prometheus_storage_inconsistent

- severity: critical
- origin: system
- summary: PROMETHEUS STORAGE INCONSISTENCY WARNING
- description: Prometheus has detected a storage inconsistency. A server restart is needed to initiate recovery.

prometheus_target_scrape_sync_too_low

- severity: warning
- origin: system
- summary: PROMETHEUS TARGET SCRAPE SYNC WARNING
- description: Prometheus target scrape sync rate is too low

promitor_azure_rate_limit_critical

- severity: critical
- origin: system
- summary: PROMITOR AZURE RATE LIMIT HIGH USAGE CRITICAL WITH THROTTLED
- description: Promitor Azure rate limit high usage, almost exhausted

promitor_azure_rate_limit_warning

- severity: warning
- origin: system
- summary: PROMITOR AZURE RATE LIMIT HIGH USAGE WARNING
- description: Promitor Azure rate limit high usage, close to be exhausted

redis_cache_too_many_connections

- severity: critical
- origin: core
- summary: Redis Cache `$labels.instance_name` too many connections
- description: Redis Cache `$labels.instance_name` has too many connections: (`$value` %)

redis_down

- severity: critical
- origin: core
- summary: Redis down (instance `$labels.instance`)
- description: Redis instance is down VALUE = `$value` LABELS = `$labels`

redis_memory_used_critical

- severity: warning
- origin: core
- summary: Redis out of configured maxmemory (instance `$labels.instance`)
- description: Redis is running out of configured maxmemory (> 90%) VALUE = `$value` LABELS = `$labels`

redis_memory_used_warning

- severity: warning
- origin: core
- summary: Redis out of configured maxmemory (instance `$labels.instance`)
- description: Redis is running out of configured maxmemory (> 90%) VALUE = `$value` LABELS = `$labels`

redis_rejected_connections

- severity: critical
- origin: core
- summary: Redis rejected connections (instance `$labels.instance`)
- description: Some connections to Redis has been rejected VALUE = `$value` LABELS = `$labels`

velero_backup_failed_total

- severity: warning
- origin: system
- summary: VELERO FAILED SCHEDULED BACKUP
- description: The latest scheduled backup `$labels.schedule` failed.

velero_backup_locked

- severity: warning
- origin: system
- summary: VELERO LOCKED SCHEDULED BACKUP
- description: The latest scheduled backup `$labels.schedule` is locked.

velero_backup_partial_failed

- severity: warning
- origin: system
- summary: VELERO PARTIALY FAILED SCHEDULED BACKUP
- description: The latest scheduled backup `$labels.schedule` partialy failed.

Logs

The Kernel Platform integrates a logging system based on Fluent-bit, Fluentd, ElasticSearch and Kibana.

Currently, **Kibana** is the official tool to manage logs in the Kernel Platform. The official Kibana User Guide is a good starting point to learn how to use Kibana.

You should not integrate third-party applications or scripts with ElasticSearch. These kind of integrations are weak because the ElasticSearch API is not part of the public interface with the OB. This means that it could change without notice for several reasons such as updating the version of ElasticSearch or changing the Kernel Platform internal architecture.

The endpoint to access to Kibana is:

> https://backoffice.$CORE.baikalplatform.com/logs

Discover

The "Discover" section in Kibana is very useful to look for logs and troubleshoot issues. You can full-text search logs using Lucene query syntax. Moreover, logs are tagged with many fields that can be useful to narrow down a search, such as:

- **kubernetes.labels.app**: name of the Kubernetes application that generated the log.
- **kubernetes.pod_name**: name of the Kubernetes pod that generated the log.
- **corr**: correlator that tracks E2E requests.
- **lvl**: log level (TRACE, DEBUG; INFO, WARN, ERROR or FATAL).

Queries that rely on a specific text are weak. The Kernel Platform can't guarantee that log messages don't change between versions. In fact, they do change. This is why metrics based on logs won't be reliable and it is not recommended to use Kibana to get metrics. Prometheus is in charge of gathering and exposing the Kernel Platform metrics and Grafana is a more suitable tool to represent them. If you think a new metric could useful, please contact the Kernel Platform Team so it can be officially included as part of the platform.

Saved dashboards, visualizations and queries are not guaranteed to be kept between Kernel Platform upgrades because all the stack, including ElasticSearch and Grafana can be upgraded to newer versions.

Logs External Forwarding Feature

Fluentd endpoint

It is possible to send Kernel Platform logs to an external system (a fluentd endpoint), too. Take into account that outbound traffic from the cloud is not free, so you might incur in additional costs if this feature is enabled.

To enable this feature you need to tune `fluentd` configuration under the `service_configs` section of your deployment config. For example:

```
service_configs:
  fluentd:
    external_forwarding:
      tls_config:
        tls_enabled: True
      remote_servers:
      - hostname:
        port:
```

Set `hostname` and `port` fields with the remote endpoint. If you configure more than one remote server, fluentd load balances the traffic to them in a round-robin order. The `hostname` value can be an IP address but it is not recommended if TLS is enabled. Turning off TLS is possible but discouraged for security reasons.

You also need to configure the vault to set a `secret_shared_key`. This key is used to verify client's identity and must be configured properly in all the remote servers. For example:

```
vault:
  fluentd:
    secret_shared_key: "mysecretkey"
```

You can find additional information regarding receivers configuration (including TLS configuration and password authentication procedure) at [https://docs.fluentd.org/v1.0/articles/in_forward].

Azure Eventhub

It is possible to send Kernel Platform logs to an external system (an azure eventhub namespace), too. Take into account that outbound traffic from the cloud is not free, so you might incur in additional costs if this feature is enabled.

To enable this feature you need to tune `fluentd` configuration under the `service_configs` section of your deployment config. For example:

```
service_configs:
  fluentd:
    external_eventhub_forwarding:
      eventhubs:
        - name: # Name Azure Eventhub forwading. Used exclusively to identify kubernetes resources associated w
ith the integration.
          endpoint: # Fully qualified domain name Azure Eventhub Namespace and port. "<Name>.servicebus.windows
.net:9093"
          connection_string: # Connection string for a Azure Eventhub Namespace. "Endpoint=sb://<Name>.serviceb
us.windows.net/;SharedAccessKeyName=<KeyName>;SharedAccessKey=<KeyValue>"
          topic: # Topic in Azure Eventhub is a Azure Eventhub Instance. Azure Eventhub Instance must be previo
usly created in Azure Eventhub Namespace.
```

Set `endpoint`, `connection_string` and `topic` fields with the remote Azure Eventhubs Namespace. Azure Eventhub Instance must be previously created in Azure Eventhub to enable this feature. Will be deployed one statefulset per remote eventhub.

Log retention

The Kernel Platform Elasticsearch has a configurable log retention time (10 days by default). After this time, logs are deleted, so they are not accessible via Kibana or Elasticsearch API.

However, the Kernel Platform creates log snapshots that are saved during a year to an Azure Storage Account. These snapshots can be restored into elasticsearch, if it is neccesary to access to older logs than the retention time. The following command restores the snapshot for the specified day:

```
$ delivery/baikops/baikops elastic snapshot restore \
    --url https://rawlogs.<core>.baikalplatform.com/api \
    --config ${CONFIG} \
    --date <YYYY-MM-DD>
```

where:

- **CONFIG**: Refers to the config name used to deploy the corresponding core, generally corresponds to `<ob>-<env>`.

Note that only one day of logs can be restored with each command execution. If you need a larger window, execute the command for all days you need.

You can execute `delivery/baikops/baikops elastic snapshot restore --help` to get more information.

Connect the local Elasticsearch cluster in Kernel Platform as a remote cluster for a external Elasticsearch.

Since GES team is part of the operation ecosystem in Kernel Platform, it has appeared a new requirement to connect the local elasticsearch cluster in Kernel Platform as a remote cluster for the elasticsearch of this team. You can connect a local cluster to other Elasticsearch clusters, known as remote clusters. Once connected, you can search remote clusters using cross-cluster search. You can also sync data between clusters using cross-cluster replication.

In this case to enable this feature you need to tune `elasticsearch` configuration under the `service_configs` section of your deployment config. For example:

service_configs: elasticsearch: retention: 10 # days before removing indexes from elasticsearch admin_password: scraper_user: "scraper" external_elastic_ip: []
external_elastic_cert: ""

Set the `external_elastic_ip`, the value is an array with validate IP/CIDR addreses. Example: ["10.10.10.10/24", "11.11.11.11/32"] to permit access. You can check the publication of the service and get the external ip address assigned to the service through the following command:

```
$ kubectl get svc -n baikal-system external-elasticsearch -o wide
```

Set the `external_elastic_cert`, the value is a string from the remote elasticsearch certificate to connect (Please use vault). It's necessary include certificates of the clusters to be connected in their truststores. It means, include Kernel Platform certificate in the remote cluster and viceversa. To get your local certificate you must execute the following command:

```
$ kubectl get secret -n baikal-system elasticsearch-es-transport-certs-public -o go-template='{{index .data "ca
.crt" | base64decode}}' > elasticsearch-share-certs-public.ca.crt
```

You will get a certificate which should be shared with the GES team to be included in its cluster. With the same procedure, the remote cluster certificate should be shared to be included in the Kernel Platform elasticsearch cluster.

Tracing

`OTEL Collector and Exporter` entities (deployed as part of Kernel Platform) are the core of Kernel tracing subsystem. They will collect and export e2e tracing info as HTTP service requests go through the system. This tracing info will be accessible by means of:

- OB tracing back-end system, as a result of enabling tracing export option.
- Kernel internal `Jaeger` tracing back-end, deployed and hosted as part of Kernel Platform.

Deployment and enablement of tracing subsystem is also configurable, so Kernel platform can be deployed without this subsystem if required (toggle set to `False`).

If tracing is enabled, as a result of this feature enablement, the following elements will be deployed:

- `OpenTelemetry Collectors`, with 2 different OTEL deployments, for collection and distribution purposes.
- `Azure Eventhub` dedicated instance, to decouple both `OpenTelemetry Collector` deployments.
- `Jaeger` back-end tracing, composed by `jaeger collector` (OTLP protocol handler) and `jaeger query` (query interface) entities.

Jaeger collector will use `ElasticSearch` from logging subsystem to store tracing data.

This tracing feature will be improved in following releases and is delivered in this realease just for the purposes of e2e distributed PoC with NOVUM in Spain.

It is important to highlight that not all requests handled by Kernel are traced, but only a percentage is traced and kept. This is controlled by sampling parameter that sets the % of requests that are sampled. Sampling decision is kept e2e so those requests marked as sampled in NOVUM will also be sampled in Kernel so e2e tracing info will be available.

Visibility of the requests tracing is possible both on OB tracing back-end or by means of Jaeger back-end deployed in the platform. The following picture presents a tracing request through APIGW in `jaeger query` ui:

You can access to Jaeger UI by navigating to the following endpoint:

> https://backoffice.$CORE.baikalplatform.com/tracing

As part of the tracing attributes, we can check the `X-Correlator` of the request, that can be used to check logs on Logging System:

Tracing deployment and sampling is configured by the following 2 config parameters:

```
enable_tracing: True

service_configs:
  tracing:
    sample_rate: 0.0001
```

`enable_tracing` toggle control in tracing feature is enabled or not. In case it is set to `false` (default value), tracing subsystem is not deployed. `sample_rate` controls the % of requests to be sampled, with default 0.01% value.

This config enable_tracing toggle must be included at both `core` and `infra` levels, with consistent values at both levels. `service _config` for tracing is only needed at `infra`.

In case it is desired to export tracing info to OB tracing back-end, the following config must be added to infra (for a cloud hosted Dynatrace tenant):

```
service_configs:
  tracing:
    external_forwarding:
      endpoint: "https://XXX.live.dynatrace.com/api/v2/otlp" # XXX: replace with OB provided
      token: !vault |  # XXX: replace with OB provided
        $ANSIBLE_VAULT;1.1;AES256
        XXX
      token_scheme: Api-Token
```

Where `endpoint` is the URL of the OTLP endpoint to be used to export OTLP traces, with the provided `token` used for authentication. The only requirement is that the `token` provided is provisioned with `Ingest OpenTelemetry traces` scope in Dynatrace tenant. Finally, `token_scheme` is the scheme that will be added in the `Authorization` header, `Api-Token` according to Dynatrace documentation.

Finally, there is an extra parameter to be set in infra config, to control the HTTP header to contain the e2e trace context (including trace-id). It is the `context` parameter, and it must be set to `w3c` for e2e tracing integration with NOVUM:

```
service_configs:
  tracing:
    context: "w3c"
```

For NOVUM tracing ingest, it is required to add the corresponding Eventhub in state config:

```
service_configs:
  eventhubs-external-tracing:
    namespace:
      sku: "Standard"
      capacity: 1
      auto_inflate_enabled: True
```

```
      maximum_throughput_units: 20
```

And, at infra level, it is required to enable reading traces from state tracing Eventhub by enabling `external_tracing` parameter:

```
tracing:
  external_tracing: True
```

Integration guide for External Services

For those external services that would need to publish traces in Kernel tracing subsystem, it is recommended to deploy locally OpenTelemetry collector as part of the external service platform.

Kernel tracing integration guideline is to use OTEL Collector contrib image as it contains latest versions of adapters required and validated with the config presented in this section.

As part of OTEL Collector pipeline definition, it is required to add a `batch` processor step to limit the number of traces per message to lower value that Azure Eventhub message limit (1MB, 500 traces with and average of 1.5KB per trace):

```
        processors:
          batch:
            send_batch_size: 500
            send_batch_max_size: 500
            timeout: 1s
```

`batch` processor will be added as last processor of the pipeline.

```
        service:
          ...
          pipelines:
            traces:
              processors: [..., batch]
              exporters: [..., kafka]
```

For the exporter step, `kafka` exporter will be used, with the following config:

```
        exporters:
          kafka:
            brokers:
            - "$BROKER"
            topic: baikal.tracing
            protocol_version: 2.0.0
            producer:
              max_message_bytes: 1000000
            auth:
              sasl:
                username: "$$ConnectionString"
                password: "$CONNECTION_STRING"
                mechanism: PLAIN
              tls:
                insecure: false
```

where `BROKER` and `CONNECTION_STRING` will be environment variables added to OTEL collector pods containing, respectively, tracing endpoint and connection string provided by Kernel platform.

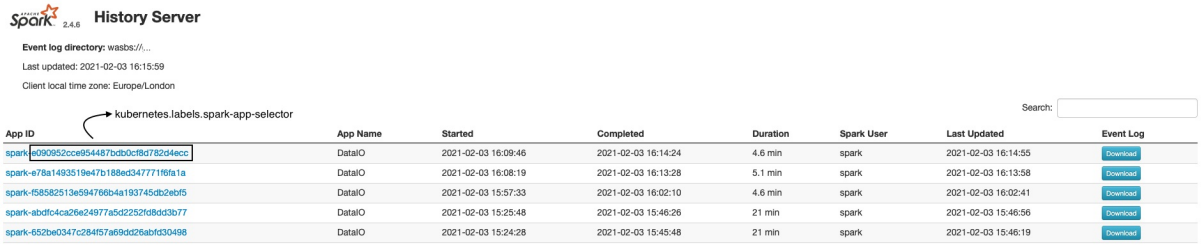
Spark History

The Kernel Platform integrates a Spark history system.

By default the algorithm execution information is stored in the spark history service account. Each `spark-xyx` file contains an algorithm execution information (Jobs, stages, timing, executors...).

ℹ️ You can navigate through the history server information at the following endpoint, authenticating the request with one of the backoffice users defined in the deployment config file (`service_configs.backoffice.users`) used during the Kernel Platform deployment:

> https://spark.$CORE.baikalplatform.com
>
> In order to find a specific id, please check the log metadata key `kubernetes.labels.spark-app-selector` in the algorithm's driver logs (any log).
>
> Note that Spark UI only shows the last N (max integer value) executions at the portal, but old executions can be routed by path directly in the URL.



Security

This section is intended to describe security recommended practices by Kernel Platform Development Team.

Kubernetes

The deployment process generates a kubeconfig file. This file contains TLS certificates, used to communicate with the Kubernetes API. Anyone with access to this file can perform any operation on the cluster (it can be considered as a root user). This is the reason why **the root kubeconfig file MUST BE KEPT IN A SAFE PLACE AND NEVER BE SHARED**. The RBAC section describes how to generate individual kubeconfig files with limited privileges.

The operations against Kubernetes API are audited. This audit logs are indexed in elasticsearch like the rest of logs of the Kernel Platform. The format of this audit logs is specified by Kubernetes, and can be found in its documentation.

Certificates

The Kernel Platform uses Let's encrypt certificates for "baikalplatform.com" subdomains, that are automatically renewed using cert-manager. TLS termination is done by the NGINX Plus Ingress.

The Kernel Platform allows the configuration of the supported TLS cipher suites and the minimum allowed TLS version, in the security section as in the following example:

```
security:
  tls_min_protocol_version: "TLSv1_2"

  tls_policy_type: "CustomV2" # required to enable TLS 1.3 support

  tls_ciphersuites:
```

```
      - "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256"
      - "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384"
      - "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256"
      - "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384"
      - "TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA"
```

The available cipher suites depend on the choosen policy type.

In general, these cipher suites are discouraged:

- use a key of less than 128-bits
- use the 3DES algorithm
- use 64-bit block ciphers
- use chain block ciphering (CBC)

However, they might be required to support legacy software.

See Azure "TLS policy overview" docs for additional information.

Azure

See the official Azure docs and `az aks rotate-certs` to rotate the cluster certificates. In case you recreate the whole infrastructure using the "parallel upgrade" procedure, all secrets will be recreated.

RBAC

RBAC (role-based access control) feature is enabled by default. This means that anyone who needs to access the cluster must have their own service accounts to operate with the cluster.

The Kernel Platform Development Team has implemented an automated tool (included in the installer) to generate kubeconfig files with the required permissions. The root kubeconfig is needed to run this tool.

Create a new kubeconfig file for a certain entity

You should generate a different kubeconfig for each config (platform operators, admins or support teams) limiting their permissions. For example, you could create a kubeconfig file for the user "mary" with a role that only grants "view" access to the cluster.

```
$ ./create-user.sh --user-name=mary --role-name=view --kubernetes-config-file=$PATH_TO_ROOT_KUBECONFIG
```

ℹ️ **view**, **edit**, **admin** and **4padmin** are the only valid roles for now.

- **admin**: Default kubernetes cluster role without access to nodes, roles and role bindings.
- **4padmin**: Customized admin cluster role with access to nodes, roles and role bindings.

Revoke grants to a certain entity

In case you may need to revoke the access to "mary", you could execute the following script to delete her grants:

```
$ ./delete-user.sh --user-name=mary --kubernetes-config-file=$PATH_TO_ROOT_KUBECONFIG
```

Cloud Secrets

The environment variables `AZURE_CLIENT_ID`, `AZURE_CLIENT_SECRET` and `AZURE_TENANT_ID` are used to deploy the platform. Once the deployment finishes, these credentials are no longer needed.

Moreover, a Service Principal is created in the Azure Active Directory. The Service Principal identifier and secret are used by some services (like velero, promitor or grafana) to access Azure Services.

The platform uses multiple Azure Storage Accounts (for algorithms, aura-conversations, backups and datasets) and deploys some services that need to access them, using an Azure access key. The keys can be regenerated in the Azure Portal (also using the CLI). The steps to rotate the keys in each case are:

- For the algorithms storage account: modify all references associated with this key in "algorithm-manager" and "rclone-config" kubernetes secrets to use a new –second– key and regenerate the original key on Azure Portal.
- For the aura-conversations storage account: The keys are used by a third-party team (Aura) to store files on the bucket so the best way to rotate them is giving Aura a new key and regenerating the first one on Azure Portal only when Aura has configured the new one.
- For the backups storage account: Used by zookeeper-backup cronjob so far. Modify parameter "AWS_SECRET_ACCESS_KEY" by the new key in "zookeeper-backup" kubernetes secret to use a new –second– key and regenerate the original key on Azure Portal.

ℹ️ The storage account for aura-conversations is legacy. It must be removed now that the Kernel Platform comes with an official API to ingest data.

How to change Azure Service Principal

See the official Azure docs.

Cluster vulnerability scans

At runtime

All Kubernetes events and Kubernetes auditing are logged and stored in ElasticSearch, to provide a security-relevant chronological set of records documenting the sequence of activities that have affected the system.

Node updates

The Kernel Platform comes with kured installed. This daemonset takes care of automatic node reboots when the need to do so is indicated by the package management system of the underlying OS. See more info about kured at the Kernel Platform services.

Docker images

Analysis of Docker images is done at development time.

The following features are in the Kernel Platform roadmap:

- Usage of signed images, coming from a private registry.

Docker private registries

The Kernel Platform Development Team owns two private Azure Container registries to store and distribute the images needed to run the platform:

- baikalalgorithms: Used in preproduction and production official environments to store the algorithms that are executed in the processing environment.

- baikalimages: Used in preproduction and production official environments to store the services' images.

Access to these private registries is based in a RBAC method to assign specific permissions to service principals: read and write for baikalalgorithms and baikaldev and read-only for baikalimages. There is one service principal for each deployed cluster and each service principal has a unique client id and client secret that can expire (1 or 5 years expiration time). Once the service principal reaches the expiration date, the process to be followed is to renew the client secret with the same value through az cli with the following command:

```
$ az ad sp credential reset --name <application_id> --years 5 --password <previous_password>
```

⚠️ If the service principal's secret is not renewed or if the password does not match the original one, the services with restarted pods will fail since they can no longer download the image from the docker registry, particularly those with `imagePullPolicy: Always`.

Network security groups

AKS configures a Network security groups (NSG) for all the nodes it manages, that allows traffic between nodes and blocks all direct traffic from the Internet. This network security group is fully managed by the AKS services, so it cannot be modified (changes will be overridden). This NSG is configured at Network Interface level, instead of subnet level.

The access to the Kernel Platform services is performed via NGINX Plus Ingress, which is exposed externally via Azure Load Balancer. By default it is configured to allow traffic from all IPs to port 443.

Network policies

A network policy is a set of network traffic rules applied to a group of pods in the Kubernetes cluster.

This is the official supported method to protect communications between services in the Kubernetes cluster, instead of Azure Network Security groups (NSG). See ./kubernetes-cluster.md#network-policies for additional info.

Azure DDoS protection

Every property in Azure is protected by Azure's infrastructure DDoS (Basic) Protection at no additional cost. If necessary, azure allows these services to be extended to a **DDoS Protection Standard plan**. Overall, it offers more debugging on the attack and support of an Azure team. The Standard plan has additional cost. The standard plan can be activated using this QNA post.

The plan automatically adjusts the thresholds that trigger the mitigation, being specific to each IP. The threshold changes in real time, for example if the traffic increases. Legitimate traffic will get through, the only way the plan will mitigate it is through rate limit.

The plan can be activated at any moment (for example when the platform is under attack).

Also, you can find additional posts with methods and procedures to try to reduce the impact of a DDoS attack at QNA, with the `ddos` tag.

Create Azure DDoS Protection Standard

The standard plan can be activated using this QNA post.

DDoS protection metrics

We can find 3 types of metrics:

- Dropped tag name (for example, Inbound Packets Dropped DDoS)
- Forwarded tag name (for example Inbound Packets Forwarded DDoS)
- No tag name (for example Inbound Packets DDoS)

The metrics can be found in Grafana in the DDoS Protection dashboard, or in Azure in the public IP metrics.

DDoS diagnostic logging

The following diagnostic logs:

- DDoSProtectionNotifications: public IP resource is under attack.
- DDoSMitigationFlowLogs: review the dropped traffic, forwarded traffic and other interesting datapoints during an active DDoS attack in near-real time.
- DDoSMitigationReports: detailed information about the attack on your resource. here will be an incremental report generated every 5 mins and a post-mitigation report for the whole mitigation period.

We can find these metrics in Elasticsearch filtering by `category` field.

Customer managed key encryption

According to this link, the creation of those Azure Storage Accounts that contain user data can be encrypted using a key directly managed by the platform.

To enable this feature, it is required to set enable_network_policies configuration parameter to true. After that, it is required to recreate the subscription (to trigger the creation of the required keyvault, key and rotation policy to be used by this feature). Once done that, the recreation of the state and core will apply the created key to encrypt the storage accounts.

To complement this feature, Key Vault audit feature is automatically activated for the Key Vault used, according to this link. These audit logs are retained during 90-days period in a dedicated Storage Account in the subscription (with `keyvaultaudit` prefix).

Web Application Firewall

As part of platform deployment Web Application Firewall (WAF) can be enabled to protect platform services exposed by NGINX Plus Ingress. This NGINX Plus Ingress WAFv4 feature is described in this link.

WAF status and execution mode is controlled by the following default infra and core configuration:

```
  security:
    waf:
      enabled: False
```

At tenant level, waf config includes the list of ingresses with WAF enabled:

```
security:
  waf:
    enabled: False
    waf_enabled_ingresses: []
```

Currently NGINX Ingress Plus WAF and l7 DoS are disabled.

Backup

This section describes the backup procedures of the Kernel Platform as well as the procedures to restore data.

Zookeeper

Zookeeper is used in the Kernel Platform to store data needed by the Kafka cluster to work. It is deployed in HA configuration and it is very resilient to failures due to its clusterized nature. Data stored by Zookeeper are written to Kubernetes persistent volumes backed up by Azure Managed Disks. This means that even in the case that the service is lost data remains in such volumes and will be there when the service is restored.

However, in case of a cloud disaster or a human error, data could be lost, and even when Kafka stores only a few KB in Zookeeper, that information is vital for Kafka, up to the point that losing the Zookeeper data would make Kafka unusable.

Backup procedure

Zookeeper data is automatically backed up on a daily basis using Burry and data is stored in a storage account whose name starts with "backups" and ends with a unique hash. A Kubernetes cronjob named zookeeper-backup takes care of it.

In case of manual backup is needed, run the following:

1. Export the environment variables:

```
$ export KUBECONFIG="<kubeconfig-file>"  # kubeconfig of the state kubernetes cluster.

# Burry only supports AWS S3 object storage, so we use a services that offers AWS S3
# interface to access Azure Storage Accounts. For this reason, the ACCESS_KEY_ID and
# the SECRET_ACCESS_KEY are the Access Keys of the storage account.
# You can find them in the Azure portal, going to the resource group "baikal-backups-rg",
# and then, at the "zookeeperXXXXX" storage account, going to the "Access keys" blade.
$ export ACCESS_KEY_ID="<storage-account-key>"
$ export SECRET_ACCESS_KEY="<storage-account-connection-string>"
```

2. Deploy minio

```
$ kubectl config set-context --current --namespace=baikal-state

$ kubectl run minio --image=minio/minio --port=9000 gateway azure --env="MINIO_ACCESS_KEY=$ACCESS_KEY_ID" -
-env="MINIO_SECRET_KEY=$SECRET_ACCESS_KEY"
```

3. Wait until minio pod is Ready and then, generate the backup. It will be saved on the storage account, in the folder `zookeeper-manual`.

```
$ export BACKUP_FOLDER="zookeeper-manual"
$ export MINIO_ENDPOINT=$(kubectl get pod minio -o json | jq -r .status.podIP)

$ kubectl run zk-backup -i --rm --image=telefonica/burry:0.4.0-1826837 -- --operation=backup --isvc=zk --en
dpoint=zookeeper:2181 --target=minio --credentials=${MINIO_ENDPOINT}:9000,ACCESS_KEY_ID=${ACCESS_KEY_ID},SE
CRET_ACCESS_KEY=${SECRET_ACCESS_KEY},BUCKET=core-${INFRA}-${CORE},PREFIX=${BACKUP_FOLDER},SSL=false
```

You can repeat the last command in case of backup failure.

1. Once backup is successfully done, run the following command to clean up the environment:

```
$ kubectl delete pod minio
```

ℹ️ The backup is done in the same cloud region the Kernel Platform is running on. It is a good practice to make a copy to another location to avoid data loss in case of a human mistake. Remember that this backup doesn't contain any sensitive data.

Restore procedure

In case of Zookeeper data loss or corruption, data must be restored from the last backup following these steps (**order is very important**). The commands must be executed using the `kubeconfig` for the State Kubernetes cluster:

1. For each kafka broker available in the cluster run the following. The `ID` is the broker identifier.

```
$ kubectl exec -t kafka-<ID> -- rm /var/lib/kafka/data/meta.properties
```

2. Stop the Kafka cluster with the following command:

```
$ kubectl scale statefulset kafka --replicas=0
```

3. Wait until the Kafka cluster has been stopped. The cluster will be stopped when there are not Kafka pods in `running` state. You can use the following command to monitor Kafka pods. The Kafka cluster could take a long time to stop (up to 10 minutes).

```
$ kubectl get pods --selector app=kafka
```

4. Stop the Zookeeper cluster if it is running, and wait for the cluster to be stopped (use `kubectl get pods --selector app=zookeeper` to monitor the Zookeeper status).

```
$ kubectl scale statefulset zookeeper --replicas=0
```

5. Remove Zookeeper´s Persistent Volume Claims:

```
$ kubectl delete pvc --selector app=zookeeper
```

6. Start the Zookeeper cluster with the following command. `N` is the original number of Zookeeper instances according to the config.

```
$ kubectl scale statefulset zookeeper --replicas=N
```

7. Start the Zookeeper cluster with the following command. `N` is the original number of Zookeeper instances according to the config.

```
$ kubectl scale statefulset kafka --replicas=N
```

8. Export the environment variables:

```
$ export KUBECONFIG="<kubeconfig-file>"  # kubeconfig of the state kubernetes cluster.

# Burry only supports AWS S3 object storage, so we use a services that offers AWS S3
# interface to access Azure Storage Accounts. For this reason, the ACCESS_KEY_ID and
# the SECRET_ACCESS_KEY are the Access Keys of the storage account.
# You can find them in the Azure portal, going to the resource group "baikal-backups-rg",
# and then, at the "zookeeperXXXXX" storage account, going to the "Access keys" blade.
$ export ACCESS_KEY_ID="<storage-account-key>"
$ export SECRET_ACCESS_KEY="<storage-account-connection-string>"
```

9. Select the backup to restore. You must look for the snapshot file in the `zookeeperXXXXX` storage account.

```
$ export SNAPSHOT_ID="<snapshot-file-name>"  # Folder in which the backup file is located
$ export BACKUP_FOLDER="<snapshot-backup-folder>"  # Name of the backup file in the cloud storage
```

10. Deploy minio

```
$ kubectl config set-context --current --namespace=baikal-state

$ kubectl run minio --image=minio/minio --port=9000 gateway azure --env="MINIO_ACCESS_KEY=$ACCESS_KEY_ID" -
-env="MINIO_SECRET_KEY=$SECRET_ACCESS_KEY"
```

11. Wait until minio is deployed, and then, restore de backup

```
$ export MINIO_ENDPOINT=$(kubectl get pod minio -o json | jq -r .status.podIP)

$ kubectl run zk-restore -i --rm --image=telefonica/burry:0.4.0-1826837 -- --operation=restore --isvc=zk --
endpoint=zookeeper:2181 --snapshot=${SNAPSHOT_ID} --target=minio --forget --credentials=${MINIO_ENDPOINT}:9
000,ACCESS_KEY_ID=${ACCESS_KEY_ID},SECRET_ACCESS_KEY=${SECRET_ACCESS_KEY},BUCKET=core-${INFRA}-${CORE},PREF
IX=${BACKUP_FOLDER},SSL=false
```

12. Once data restore is successfully done, run:

```
$ kubectl delete pod minio
```

13. Stop and start the Kafka cluster for the restoration to take effect.

ℹ️ If any topics were created after the latest backup, simply recreate the topics and data will be there.

Kafka

Kafka is deployed in HA configuration and it is very resilient to failures due to its clusterized nature. Data stored by Kafka goes to Kubernetes persistent volumes backed up by Azure Managed Disks.

This means that even in the case that the service is lost, data remains in such volumes and will be there when the service is restored. But in the case of a cloud disaster or a human error, data could be lost.

Backup procedure

Kafka backup is performed by a cronjob named `kafka-backup` that stores the messages of selected topics and consumer groups' offsets in an external cloud storage. Although Kafka servers are deployed in the State Kubernetes cluster, the job is launched in the Services Kubernetes cluster, because topics are managed by the Kernel Platform Core services. Each time the cronjob is executed it creates a new folder whose name is a timestamp in the blob container which is located into the `kafka` storage account in the `baikal-backups-rg` resource group. Each backup consists of:

- Topics backup: a copy of each message stored in each partition of each topics being backed up. They are stored in the `topics` folder, where there is a folder for each topic, and inside that folder, a couple of files per partition:
  - `<topic_name>-<partition>` : contains a raw dump of each message, including key and payload.
  - `<topic_name>-<partition>.json` : contains a JSON formatted dump of each message, including topic, partition, offset, timestamp, timestamp type, broker, key and payload.
- Consumer groups' offsets backup: in the `offsets` folder there is a JSON formatted file for each consumer group which contains the offset and lag for each partition of each topic such a consumer group was subscribed to at the time the backup was done.

Kafka backup can be configured through the `backup.kafka` settings in the config:

- `enabled` : whether the Kafka backup is enabled or not.
- `schedule` : crontab spec used to schedule the Kafka backup cronjob.
- `retention_days` : number of days the backups will be retained in the storage account before being deleted.

Restore procedure

In the case of Kafka data loss, data must be restored by republishing messages from the cloud storage to its corresponding topic and partition. Note that topics to be restored must exist before republishing messages.

The recommended procedure to republish messages is based on `azcopy` and `kafkacat` tools, which are available in the Docker image `telefonica/kafka-toolbelt-cloud`. Follow these steps to republish the messages of a partition:

⚠️ You must have configured the `kubeconfig` for the Services Kubernetes cluster.

1. Run a pod in the core namespace of the cluster using the former Docker image.

```
$ kubectl run kafka-recovery -it --rm --image=telefonica/kafka-toolbelt-cloud --namespace baikal-${CORE}
```

2. Locate the Kafka backup storage account in Azure Portal and set its name, key and blob container name in environment variables:

```
$ KAFKA_BACKUP_ACCOUNT_NAME="kafkaca1df3ec03d21ca5f1e"
$ KAFKA_BACKUP_ACCOUNT_KEY="VWY8f1BJ5P...W4MI16xSz5w=="
$ KAFKA_BACKUP_CONTAINER_NAME="core-<cluster>-<core>"
```

3. Get a SAS token to let `azcopy` to gain access to the backup storage.

```
$ SAS_TOKEN_EXPIRY=$(date -u -d "8 hours" '+%Y-%m-%dT%H:%MZ')
$ SAS_TOKEN=$(az storage account generate-sas \
              --account-name ${KAFKA_BACKUP_ACCOUNT_NAME} \
              --account-key ${KAFKA_BACKUP_ACCOUNT_KEY} \
              --services b --resource-types o --permissions r \
              --expiry ${SAS_TOKEN_EXPIRY} --https-only | tr -d \")
```

4. Restore messages for the partition `0` of the topic `baikal.core.algorithms` :

```
$ BACKUP_DATE="2021-02-16T12:24:18Z"  # Timestamp of the backup to be restored
$ TOPIC="baikal.core.algorithms"
$ PARTITION=0
$ azcopy copy https://${KAFKA_BACKUP_ACCOUNT_NAME}.blob.core.windows.net/${KAFKA_BACKUP_CONTAINER_NAME}/${BACKUP_DATE}/topics/${TOPIC}/${TOPIC}-${PARTITION}?${SAS_TOKEN} --from-to BlobPipe | \
    kafkacat -P -b kafka.baikalplatform.local:9092 -t ${TOPIC} -p ${PARTITION} -K "|" -q
```

This procedure must be repeated for each partition to be restored. If you want to restore all the topics and partitions, you can use the following `baikops` command:

```
$ baikops kafka restore-topics --infra ${INFRA} --core ${CORE} --last-backup
```

Your can follow the progress of the restore procedure by accessing to the logs of the pod launched by this script:

```
$ kubectl logs -f kafka-restore-<xxxx>

Getting a SAS token...
Done

####################
# Restoring topics #
####################
```

```
Restoring topic 'baikal.core.algorithms', partition 0...
Done

Restoring topic 'baikal.core.algorithms', partition 1...
Done

Restoring topic 'baikal.core.algorithms', partition 2...
Done
```

Before running the `restore-topics.sh` script, ensure that your `kubectl` is using the right Kubernetes context pointing to the core namespace.

⚠️ Be aware that restoring the content of the topics do not restore the consumer groups' offsets. Bear in mind that each message's offset may have changed after the restore procedure.

Setting consumer groups' offsets

If you need to modify consumer groups' offsets, use the `kafka-consumer-groups` tool, also available in the Docker image `telefonica/kafka-toolbelt-cloud`. This tool supports many ways of setting offsets per topic, partition, to the earliest/latest offset, to a particular offset, to the nearest offset to a timestamp, etc. Run `kafka-consumer-groups --help` to see all the supported options. See below some typical scenarios:

Set all the consumer groups' offsets to the earliest one for each topic/partition:

```
$ kafka-consumer-groups --bootstrap-server kafka.baikalplatform.local:9092 --all-groups --all-topics --execute
--reset-offsets --to-earliest
```

Set the offsets of a specific consumer group to the latest one for a specific topic:

```
$ kafka-consumer-groups --bootstrap-server kafka.baikalplatform.local:9092 --group <consumer-group-name> --topi
c <topic-name> --execute --reset-offsets --to-latest
```

Set the offsets of a specific consumer group to a specific offset for a specific topic and partition:

```
$ kafka-consumer-groups --bootstrap-server kafka.baikalplatform.local:9092 --group <consumer-group-name> --topi
c <topic-name>:<partition> --execute --reset-offsets --to-offset <offset>
```

When to modify consumer offsets

The main thing to understand when restoring data to Kafka is that the offset of the restored messages will most likely not match the offset of the original messages. This happens because Kafka performs data cleanup on the original topics.

All of the topics that have "compaction" enabled are replayable and idempotent. This means that you can ignore setting the consumer offsets and the platform will re-process all of the messages. However, as part of the recovery process the topic lag should be monitored until it has disappeared.

Even if technically you could ignore the offsets for compacted topics, that might not always be a feasible option. For example, the consent topic might have millions of records which would need to be reprocessed, which might take a long time. In order to reduce the amount of reprocessing required, you might want to reset the offset a point in time where you are sure that every previous message had been processed. For example, it is a safe bet to assume that any messages that were produced one hour prior to the backup had already been processed.

In order to do that you need to:

1. Decide on a cutoff timestamp.

2. Go to the JSON Kafka backup files for each topic-partition and look for the last message that happened before the cutoff timestamp (by checking the `ts` field).
3. Search for that message in the restored Kafka topic, to understand what the new offset is.
4. Use the `kafka-consumer-group` command to set that offset for any consumer group that is reading that topic-partition.

What about non-compacted topics? Non-compacted aren't replayable. For example, replaying the notifications topic would re-send all of the notifications again. For these topics, it is better to set the offset to latest in all of the consumers. There is a small chance that some messages won't be processed, but that loss should be admissible in the event of a disaster.

Restarting Kafka Producer services

In order to ensure that services that produce messages to Kafka keep working correctly, they must be restarted.

The safest way to perform this restart is to do it in the following way:

1. Restart `primary` producers:

```
for deployment in $(kubectl get deployments -l kafka-primary-producer=true -o name); do
  kubectl rollout restart $deployment
done
```

2. Wait for `primary` producers to be restarted:

```
for deployment in $(kubectl get deployment -l kafka-primary-producer=true -o name); do
  kubectl rollout status $deployment
done
```

3. Restart `secondary` producers:

```
for deployment in $(kubectl get deployments -l kafka-secondary-producer=true -o name); do
  kubectl rollout restart $deployment
done
```

4. Wait for `secondary` producers to be restarted:

```
for deployment in $(kubectl get deployment -l kafka-secondary-producer=true -o name); do
  kubectl rollout status $deployment
done
```

Database

Backup procedure

The Kernel Platform databases rely on Azure PostgreSQL service which provides an automated Backup procedure described here.

Taking advantage of this backup feature and point-in-time restore capability, we have designed and implemented an automatic procedure to restore any of the Kernel Platform database instances (core) and switch all the Kernel Platform services to the new instance.

Restore procedure

Following guide is intended to be used in case of data corruption, partial or total data loss but assuming the original database instance existence.

Prerequisites

Before running any of this guide steps, you need to export the Azure credentials used to gain access to Azure services.

```
$ export AZURE_CLIENT_ID=xxx
$ export AZURE_CLIENT_SECRET=xxx
$ export AZURE_SUBSCRIPTION_ID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx
$ export AZURE_TENANT_ID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx
```

This user must have:

- the `Azure Kubernetes Service Cluster Admin Role` and `Contributor` roles assigned in the Azure subscription (Access Control / IAM).

Database point-in-time restore

You must use the Kernel Platform installer, that includes the `baikops` tool. Change the directory to the installer directory and execute:

```
$ ./baikops database restore --state ${STATE} --restore-point-in-time <the point in time to restore from (ISO86
01 format)>
```

Review https://docs.microsoft.com/en-us/cli/azure/postgres/server?view=azure-cli-latest#az_postgres_server_restore for further information.

Services switching

As in the database restore operation, you can use `baikops` to switch the service configuration to point to the new database.

```
$ ./baikops database switch-services --state ${STATE} --infra ${INFRA} --core ${CORE}
```

⚠️ state-create must be launched to reconcile the database configuration.

Kubernetes

Backup procedure

As described in services guide, we use Velero as a backup/restore tool for our Kubernetes clusters.

We have implemented a full cluster backup using Velero Scheduled Backups that allows you to back up your data at recurring intervals. The first backup is executed when the schedule is created, the following backups are triggered at the schedule's specified interval. These intervals are specified by a Cron expression.

Scheduled backups are saved with the name `<SCHEDULE NAME>-<TIMESTAMP>`, where `<TIMESTAMP>` is formatted as `YYYYMMDDhhmmss`.

This procedure can be customized through the deployment config in the `backup` section for the Services Kubernetes cluster, and in the `state.backup` section for the State Kubernetes cluster.

```
#
# Backup configuration
#
backup:
  kubernetes:
    enabled: True
```

```
    schedule: "0 3 * * *"  # Every day at 3:00
    include_volumes: False
    retention_days: 30
```

Scheduled backups could be enabled or disabled (this change could be only applied in state and infra creation stage). You could also change the schedule frequency (how often the backups are performed) and whether you want to include PersistentVolumes snapshot as part of the backup. Finally, the backup retention parameter is used to determine the persistence period of the backup information before to be removed from the backup storage.

Restore procedure

This guide provides information regarding the assets to be used in case of data corruption, partial or total data loss in the cluster.

Prerequisites

Before executing any of the steps in this manual, you need to export the Kubernetes config file (kubeconfig), used to gain access to the Kernel Platform Kubernetes cluster. Depending on which cluster you want to recover, you must configure the Services Kubernetes cluster or the State Kubernetes cluster.

```
$ export KUBECONFIG=<kubeconfig-file>
```

Also, you need to export the Azure credentials used to gain access to Azure services.

```
$ export AZURE_CLIENT_ID=xxx
$ export AZURE_CLIENT_SECRET=xxx
$ export AZURE_SUBSCRIPTION_ID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx
$ export AZURE_TENANT_ID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx
```

This user must have:

- the `Azure Kubernetes Service Cluster Admin Role` and `Contributor` roles assigned in the Azure subscription (Access Control / IAM).

Partial recovery

You could have deleted or corrupted Kubernetes resources from specific Velero backup which are located in the `baikal-backups-rg` resource group and re-apply them using `kubectl`.

Full cluster recovery

⚠️ baikal-infra and baikal-system namespace must exist with velero and aad-pod-identity deployments. Run deployment-infrastructure to restore those services.

In case of total data loss, you can recover a complete Velero backup and restore/validate data integrity in Persistent Volumes. Use the following procedure to restore a Velero backup previously scheduled:

```
$ export KUBECONFIG=/path/to/kubeconfig/file

$ velero client config set namespace=velero
$ velero get backups

# Select the name and timestamp of the backup to be used in restore operation and execute the following command:

$ VELERO_BACKUP_NAME="YOUR_SELECT"

# Vars needed to clean/restore:
```

```
$ EXCLUDED_NAMESPACES=(baikal-infra baikal-system calico-system default gatekeeper-system kube-node-lease kube-public kube-system tigera-operator velero)
$ EXCLUDED_RESOURCES="hpa,mutatingwebhookconfigurations,orders,scaledobjects.keda.sh"
$ NAMESPACES=$(kubectl get namespaces -o name | cut -d'/' -f 2)
$ FILTERED_NAMESPACES=$(echo ${NAMESPACES[@]} ${EXCLUDED_NAMESPACES[@]} | tr ' ' '\n' | sort | uniq -u)

# Define restore names:
$ VELERO_RESTORE_NAME=RESTORE-${VELERO_BACKUP_NAME}

$ velero restore create ${VELERO_RESTORE_NAME} --from-backup ${VELERO_BACKUP_NAME} --exclude-namespaces ${EXCLUDED_NAMESPACES} --exclude-resources  ${EXCLUDED_RESOURCES} -w

# Delete undesired namespaces:
$ NAMESPACES_TO_DELETE=$(kubectl get namespaces -l execution-id -o name | cut -d'/' -f 2)
$ for NAMESPACE in ${NAMESPACES_TO_DELETE[*]}; do
>   echo -e "\tDeleting namespace '${NAMESPACE}' ..."
>   kubectl delete namespace ${NAMESPACE} --ignore-not-found
> done

# Download Velero backup:
$ velero backup download ${VELERO_BACKUP_NAME}
$ mkdir -p ${VELERO_BACKUP_NAME}
$ tar xf ${VELERO_BACKUP_NAME}-data.tar.gz --directory ${VELERO_BACKUP_NAME} resources/jobs.batch/namespaces

# Reapply jobs:
$ JOBS_PATH="${VELERO_BACKUP_NAME}/resources/jobs.batch/namespaces"
$ NAMESPACES=$(ls ${JOBS_PATH} | xargs)
$ EXCLUDED_NAMESPACES_REGEX=$(echo ${EXCLUDED_NAMESPACES//,/|})
$ for NAMESPACE in ${NAMESPACES[*]}; do
>   set +e
>   FILTERED_NAMESPACE=$(echo ${NAMESPACE} | egrep -v "${EXCLUDED_NAMESPACES_REGEX}")
>   set -e
>   if [ -n ${FILTERED_NAMESPACE} ]; then
>     echo "NAMESPACE: ${NAMESPACE}"
>     INCLUDED_JOBS=($(grep -irwlv "job-cron" ${JOBS_PATH}/${NAMESPACE}))
>     for FILE in ${INCLUDED_JOBS[*]}; do
>       cat ${FILE} \
>         | jq 'del(.spec.selector) | del(.spec.template.metadata.labels["controller-uid"]) | del(.spec.template.metadata.labels["batch.kubernetes.io/controller-uid"]) | del(.metadata.uid)' \
>         | kubectl -n ${NAMESPACE} apply -f -
>     done
>   fi
> done

# Reapply ScaledObjects:
$ SCALEDOBJECTS_PATH="${VELERO_BACKUP_NAME}/resources/scaledobjects.keda.sh/namespaces"
$ NAMESPACES=$(ls ${SCALEDOBJECTS_PATH} | xargs)
$ EXCLUDED_NAMESPACES_REGEX=$(echo ${EXCLUDED_NAMESPACES//,/|})
$ for NAMESPACE in ${NAMESPACES[*]}; do
>   set +e
>   FILTERED_NAMESPACE=$(echo ${NAMESPACE} | egrep -v "${EXCLUDED_NAMESPACES_REGEX}")
>   set -e
>   if [ -n ${FILTERED_NAMESPACE} ]; then
>     echo "NAMESPACE: ${NAMESPACE}"
>     INCLUDED_SCALEDOBJECTS=($(grep -irwl "ScaledObject" ${SCALEDOBJECTS_PATH}/${NAMESPACE}))
>     for FILE in ${INCLUDED_SCALEDOBJECTS[*]}; do
>       cat ${FILE} \
>         | kubectl -n ${NAMESPACE} apply -f -
>     done
>   fi
> done

# Disable algorithm streams:
$ ./baikops job algorithm-streams \
   --infra ${INFRA} \
   --core ${CORE} \
   disable

# Enable the previous algorithm streams:
```

```
$ ./baikops job algorithm-streams \
   --infra ${INFRA} \
   --core ${CORE} \
   enable

# Only wait to stabilization
```

Review Velero disaster recovery documentation for further information.

Full state cluster recovery

⚠️ velero namespace must exist with velero deployments and baikal-infra namespace must exist with aad-pod-identity deployment. Run deployment-state to restore those services and delete "baikal-state" namespace so you can recover it from the backup.

In case of total data loss, you can recover a complete Velero backup and restore/validate data integrity in Persistent Volumes. Use the following procedure to restore a Velero backup previously scheduled:

```
$ export KUBECONFIG=/path/to/kubeconfig/file

$ velero client config set namespace=velero
$ velero get backups

# Select the name and timestamp of the backup to be used in restore operation and execute the following command:

$ VELERO_BACKUP_NAME="YOUR_SELECT"

# Vars needed to clean/restore:
$ EXCLUDED_NAMESPACES=(baikal-infra calico-system default kube-node-lease kube-public kube-system gatekeeper-sy
stem velero)
$ EXCLUDED_RESOURCES="orders,mutatingwebhookconfigurations,jaegers.jaegertracing.io,opentelemetrycollectors.ope
ntelemetry.io"
$ NAMESPACES=$(kubectl get namespaces -o name | cut -d'/' -f 2)
$ FILTERED_NAMESPACES=$(echo ${NAMESPACES[@]} ${EXCLUDED_NAMESPACES[@]} | tr ' ' '\n' | sort | uniq -u)

# Define restore names:
$ VELERO_RESTORE_NAME=RESTORE-${VELERO_BACKUP_NAME}

$ velero restore create ${VELERO_RESTORE_NAME} --from-backup ${VELERO_BACKUP_NAME} --exclude-namespaces ${EXCLU
DED_NAMESPACES} --exclude-resources  ${EXCLUDED_RESOURCES} -w

# Delete undesired namespaces:
$ NAMESPACES_TO_DELETE=$(kubectl get namespaces -l client-id -o name | cut -d'/' -f 2)
$ for NAMESPACE in ${NAMESPACES_TO_DELETE[*]}; do
>   echo -e "\tDeleting namespace '${NAMESPACE}' ..."
>   kubectl delete namespace ${NAMESPACE} --ignore-not-found
> done

# Only wait to stabilization
```

Kernel Platform Administration Guide

The Administration guide describes how the Kernel Platform Service can be administered through the different available commands and tools. It is intended to provide useful information for the Kernel Platform administrators.

In this guide we will go through each of the concepts that the Kernel Platform exposes which need to be administered.

The Kernel Platform provides two administration interfaces: a web portal and an Admin API. The portal is more user-friendly and therefore it will be the main focus of this guide. However, the portal actually uses the Admin API under the scenes and all the operations and parameter explained in this guide are applicable to the API as well. If you want to use the API instead, please take a look at Admin API and QnA.

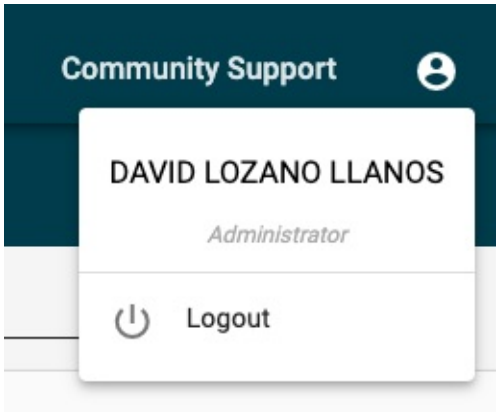
The following sections show the pre-requisites needed to use each administration interface.

Accessing the Admin Portal

The biggest pre-requisite to use the administration portal is to have your email in the administrator email list. This list is defined at deployment time. Check with the operations team that your Telefonica email address is configured within the `service_configs.portal_backend.admin_emails` property of the deployment config.

Open up the Admin Portal at `www.${CONFIG}.baikalplatform.com` and log in with your Office365 credentials.

Once logged in, on the top-left corner you will see a user icon. Click on it and a card will fold down, including your name as well as your user type. You will see the word "Administrator" if you are one of them.



If you are logged in as a normal user, that means that your email address isn't listed as one of the administrators.

Overview of main concepts

The Kernel Platform is a platform that provides capabilities to perform end-user authentication, secure API exposition as well as a standardized data lake to store huge amounts of information and then be able to run big data algorithms using that data. In this regard, as a Kernel Platform administrator, you will need to configure and administer concepts around three main fields:

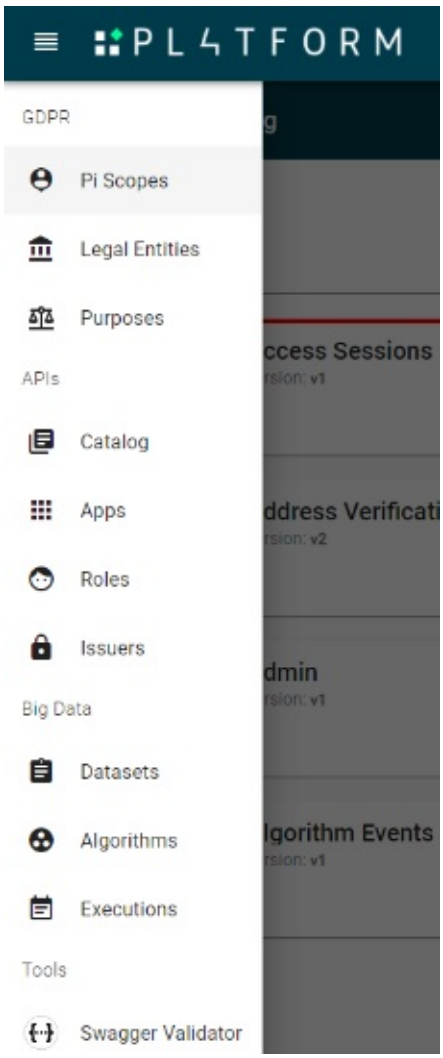
- **API exposition and consumption**: what APIs are exposed, how they are protected, which applications can consume them and how.
- **Big data repository and algorithms**: register different standard Datasets and their versions, read and/or write data of these Datasets with an application, and set up algorithms that can be executed within the Kernel Platform reading and/or writing data of these Datasets.
- **GDPR**: The GDPR is a data protection regulation approved by the EU. One of the keystones of the Kernel Platform is

controlling the access to Personal Information through any kind of interfaces, i.e. and API or a Dataset. To do so, the Kernel Platform uses different concepts (purposes, pi-scopes and consents) that will need to be managed by administrators.

The next sections deal with the administration of the concepts around these three main fields. It starts by explaining how to manage GDPR concepts as it commonly applies both for APIs and Bid Data capabilities.

Browsing the portal

The main tool to browse to the different Admin Portal sections is to use the menu in the upper left corner, as shown in the next image.

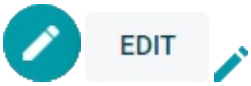
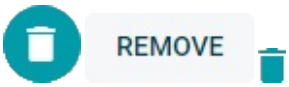


Common controls

The Admin Portal uses a set of icons or buttons to express common operations on the different manageable concepts.

Create a new entity (e.g. new Application, API, etc) is expressed by the following icon:

Download, edit and delete an entity (e.g. new Application, API, etc) can be achieved by clicking on the following icons:

- Download 
- Edit 
- Delete 

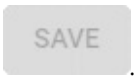
Cancel form data or save edit or new form data (e.g. new Application, API, etc) can be achieved by clicking on the following icons:

Rotate encryption key can be achieved by clicking on the following icons:

Required fields

All required fields are marked with * at the end of the field label:

If the form is not completely valid, the 

will appear disabled 

.

Complementary information

In addition to this guide, you will find interesting information on the following links that may help you get a deeper knowledge and understanding of how the Kernel Platform works:

- Developer Guides. We encourage you to start by reading the definition of API basic concepts and will point you to other sections throughout this guide in order to help you better understand.
- QnA

GDPR

The GDPR is a data protection regulation approved by the EU. This regulation specifies a set of requirements for companies that handle Personal Information (PI). From 25th May 2018, companies are required to adhere to this new regulation.

The Kernel Platform has been designed to fully support and comply with the spirit and letter of this regulation, which provides a great level of protection for our customers.

Please note that this regulation only applies to PI. Anonymized data and data which is inherently non-PI (e.g. cell information) does not need to meet GDPR requirements.

The GDPR introduces several key concepts, which it builds upon to define how to handle data. These core concepts are the following:

- Data processing: storing, transforming or accessing PI is considered processing data.
- Purpose: The reason for wanting to process PI. For example, an Application might want to handle PI to create a movie recommendation for a customer.
- Consent: an explicit, opt-in action that the customer takes to allow processing of her Personal Information. Consent can be a signature on a paper, a voice recording or clicking on an "Authorize" button on a website. The Kernel Platform offers the mechanism to gather and manage user consents from Applications.

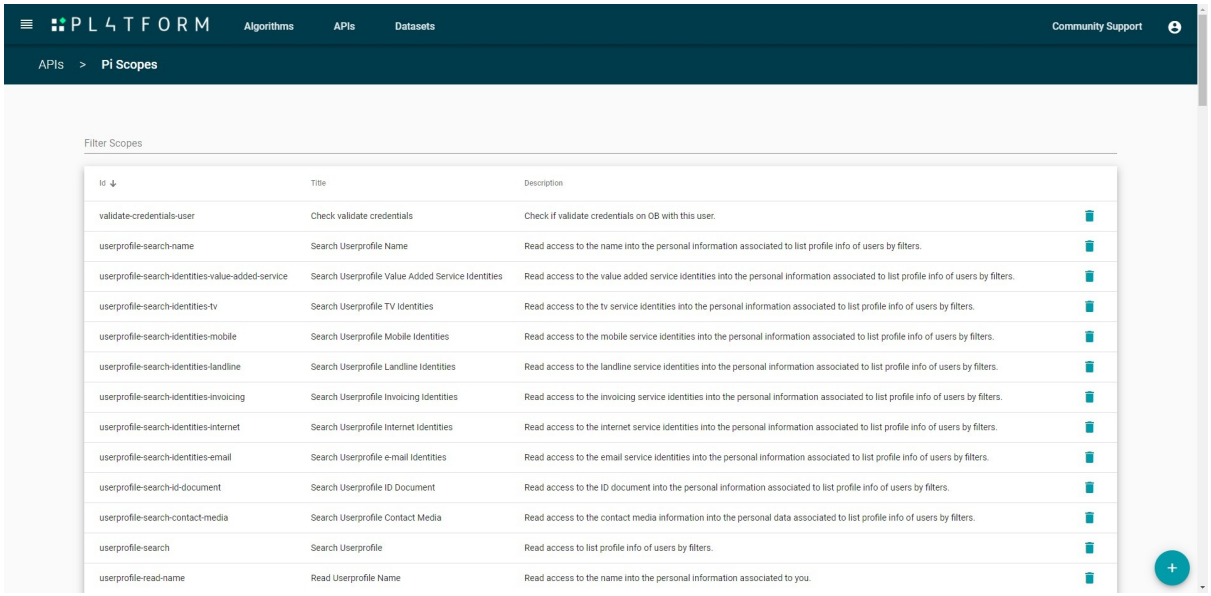
PI-Scopes

The Kernel Platform applies the concept of `scope` to label information resources and the applicable operations on these resources. Thus, a scope may refer to an HTTP API endpoint plus HTTP verb combination or to a Dataset plus the type of access (read or write). Scopes are useful to control the access to resources as they are used to list what each consuming entity, i.e. Applications or Algorithms, is allowed to do.

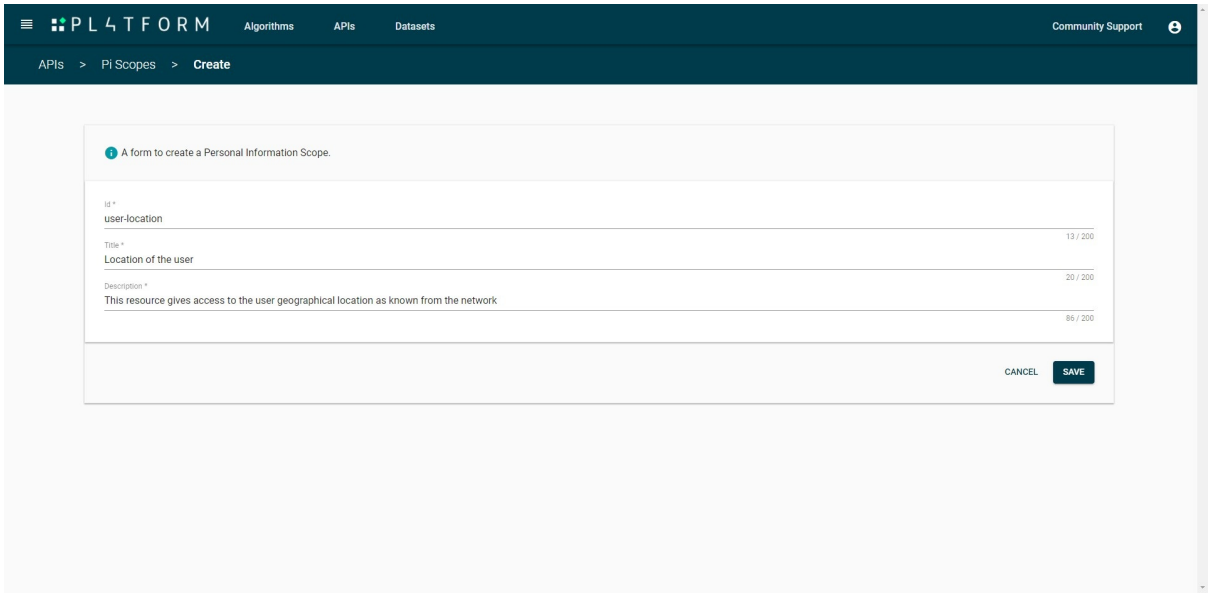
When the resource contains personal information, then we refer to it as a Personal Information Scope, in short, a PI-Scope. For example, the capability to GET the history of invoices of a user through the Invoicing API maps to the `invoices-read` PI-Scope while the capability to write the In-Application Browsing history of a given user maps to the `xxxx` PI-Scopes.

PI-Scopes need to be registered in the platform before the actually exposing the information resources.

To get the list of already registered PI-Scopes, please go to the PI Scopes sections and you will find something like the following. From that list you can also edit and delete PI-Scopes.



To create a new one, click on  at the bottom of the page and fill in a form like the following:



- **Id**: used internally to identify the resource.
- **Title**: brief description.
- **Description**: extended description.

⚠️ Note that title and description can be used by Applications and portals in the UX to inform final users when managing their consents, so make sure it is understandable by regular end-users and complies with business needs.

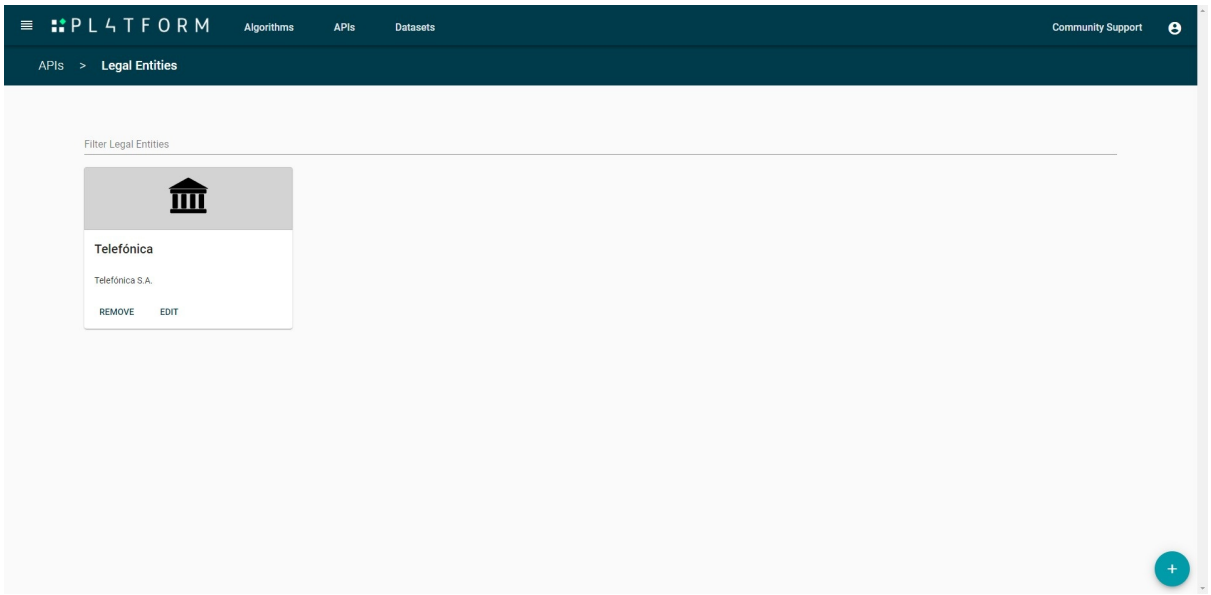
A PI-Scope can't be edited once created.

A PI-Scope can be deleted cliking on  .

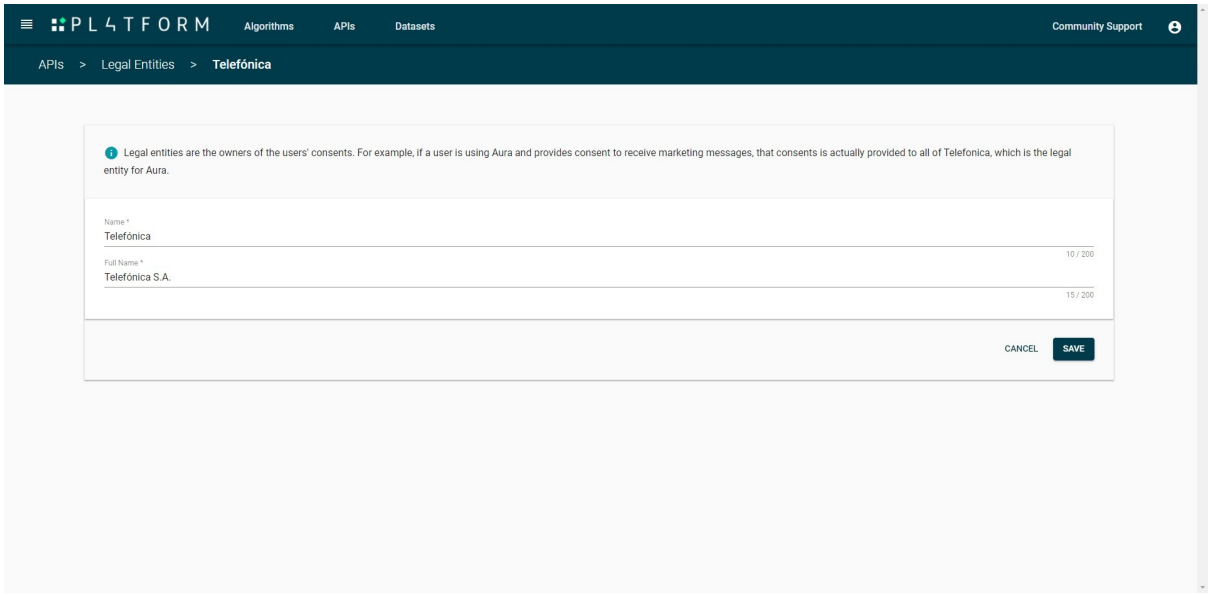
Legal Entities

When a users grant consents, they do so for Legal Entities to access/process their Personal Information under a specific purpose. Legal Entities are the subject that receives the consents. There can be multiple legal entities. Typically they will be different Telefónica brands, for example, Movistar, O2 or Vivo.

To get the list of already registered Legal entities, please go to the Legal Entities sections and you will find something like the following. As you can see, from the cards, you can also remove and edit existing entries.

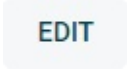


To create a new one, click on  at the bottom of the page and fill in a form like the following.

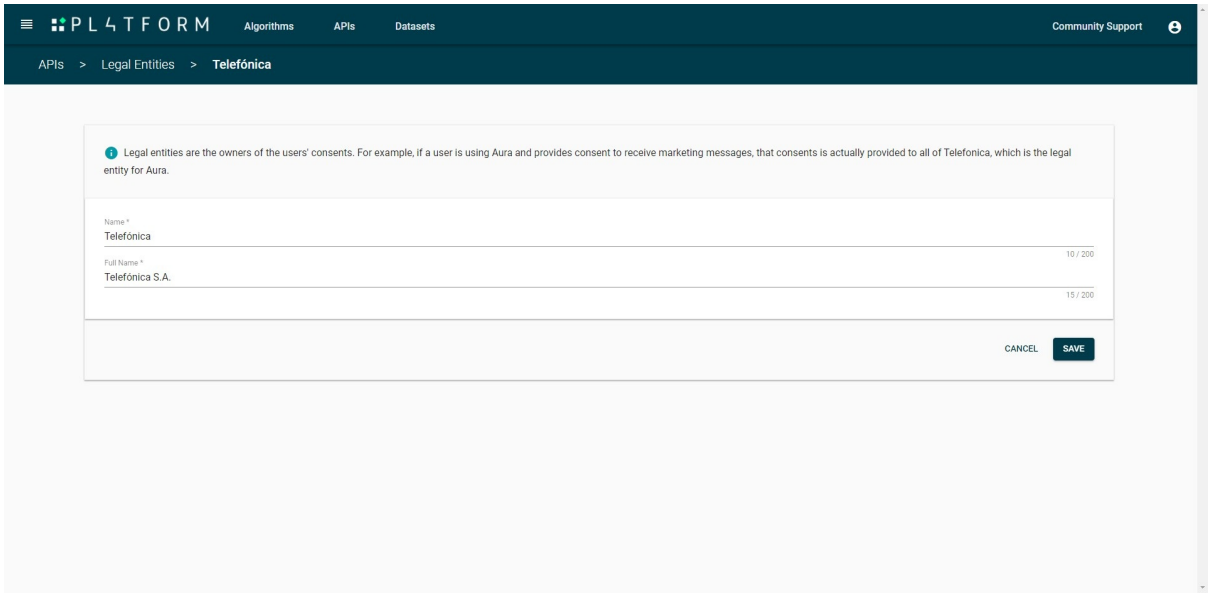


- **Name**: short name to identify the legal entity.
- **Full Name**: complete full name to identify the legal entity.

⚠️ Note that name and full name can be used by apps and portals in the UX to inform final users when managing their consents, so make sure it is understandable by regular end-users and complies with business needs.

To edit one, click on 

and it can edit the same information than in create form.

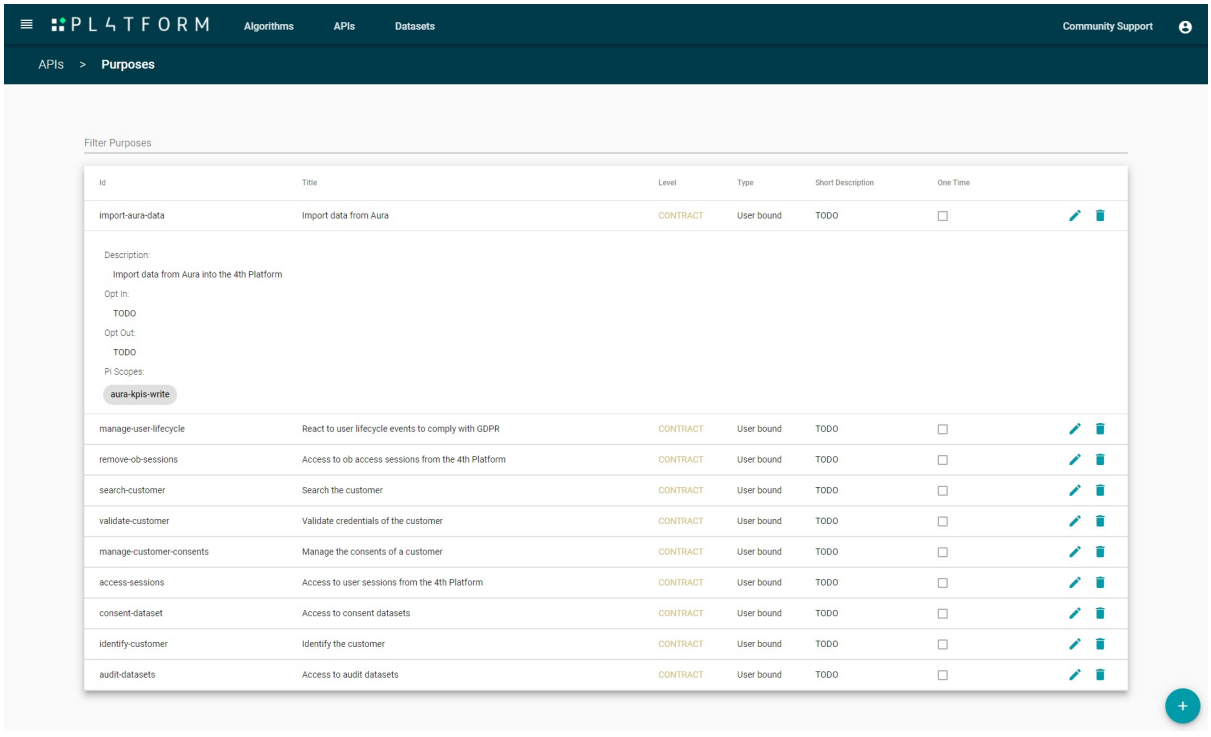


A Legal Entity can be deleted cliking on REMOVE.

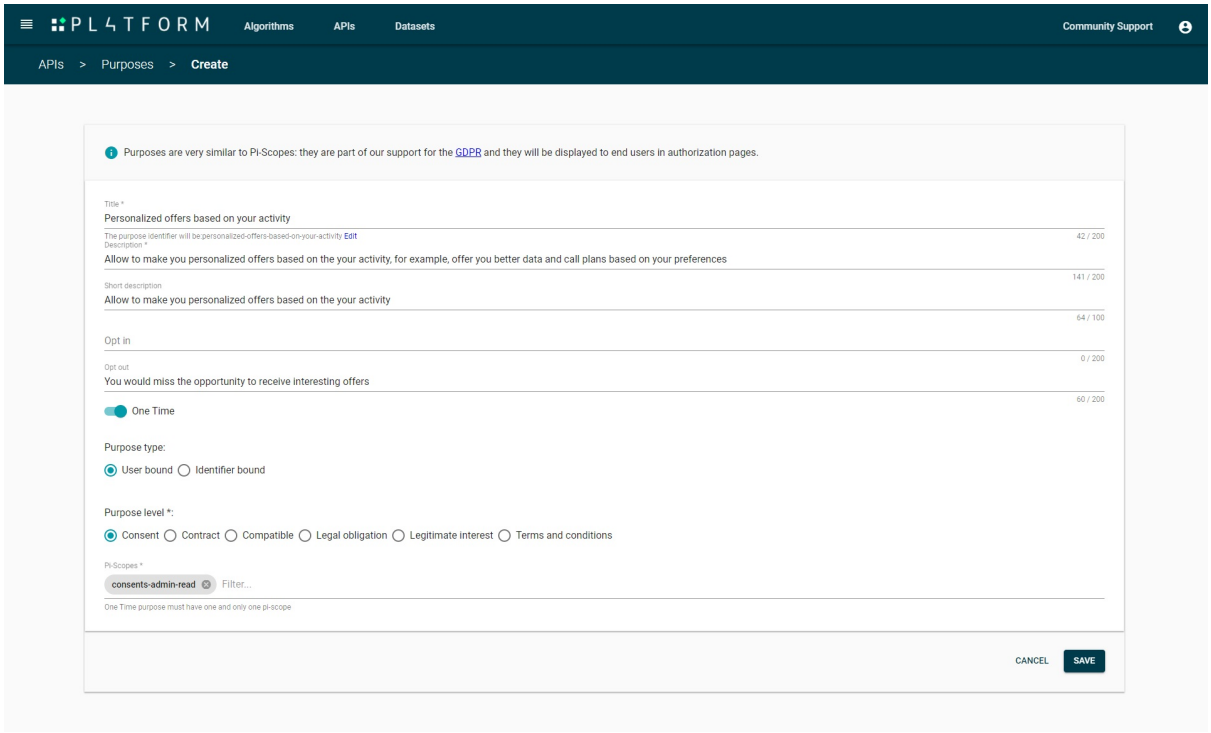
Purposes

A purpose declares what the Application or Algorithms intends to do with a set of Personal Information resources. As such, it provides a human understandable description of why the data processing takes place and a list of PI-Scopes giving access to Personal Information processing. To read further details on purposes, their types and the different available levels, please refer to https://developers.baikalplatform.com/docs/latest/consents/purposes.html.

To get the list of already registered Legal entities, please go to the Legal Entities sections and you will find something like the following. From that list you can also edit and delete purposes.



To create a new one, click on  at the bottom of the page and fill in a form like the following.

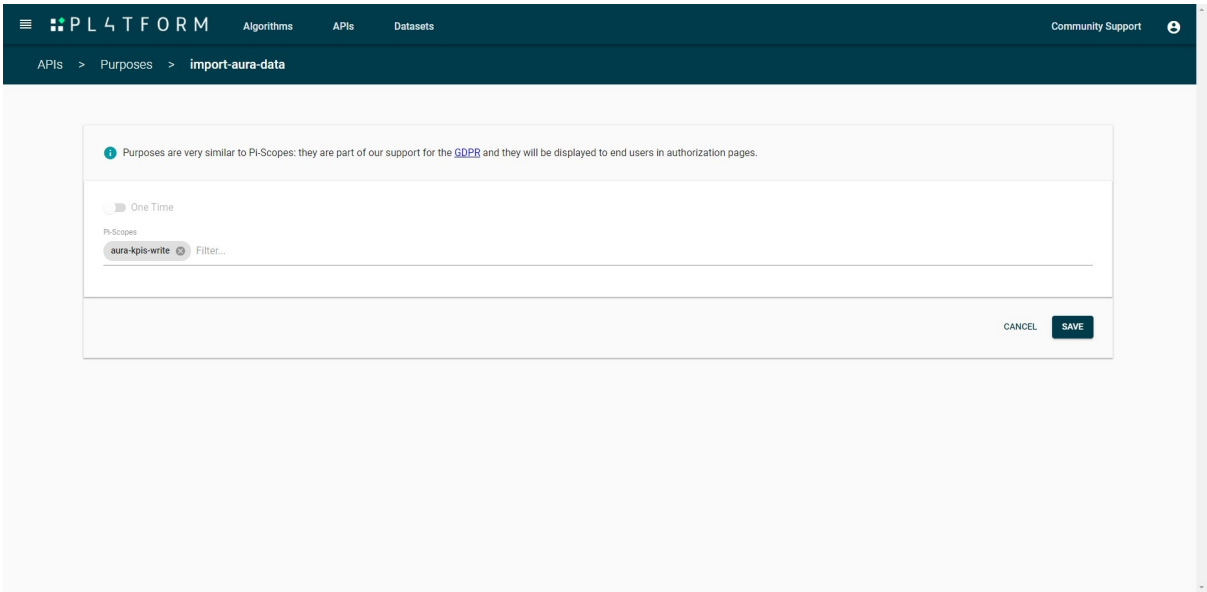


- **Title**: self-descriptive title of the purpose.
- **Description**: complete description of what the Application or Algorithm intends to do with the Personal Information under consideration.
- **Short description**: a shorter version of the description to be used when UX limits the amount of information that can be displayed to the final user.
- **Opt-in**: describe here what the user would get by granting consent.
- **Opt-out**: describe here what the user would miss by revoking consent.
- **One time**: Identify the purpose as a one-time use. All tokens requested with this purpose can only be used once. This purpose can only have one PI-scope assigned by design.
- **Purpose type**: whether the purpose applies to all the identifiers of a given user (user-bound) or only to one of them (identifier-bound).
- **Purpose level**: purpose level as specified by the GDPR law.
- **PI-Scopes**: list of Personal Information Resources the user would provide access to when granting consent for this purpose.

⚠ Note that title, description, short description, opt-in and opt-out can be used by Applications and portals in the UX to inform final users when managing their consents, so make sure it is understandable by regular end-users and complies with business needs.

⚠ One time purpose must have one and only one PI-scope.

To edit one, click on  and it can edit the following information:



- **PI-Scopes**: list of Personal Information Resources the user would provide access to when granting consent for this purpose.

⚠️ Note that a purpose only can be edited if it has a "Contract" or "Legal obligation" purpose level.

⚠️ One time purpose must have one and only one PI-scope.

A Purpose can be deleted cliking on 🗑.

Consents

A consent is the "document" that grants a legal entity (e.g. Telefonica or a specific 3rd Party) access to a set of PI scopes of a given user, under a specific purpose. Consent can be issued for the user as a whole, and thus be associated to a user_id, or for a specific user identifier (e.g. associated to a concrete user's phone number), as dictated by the purpose_type.

A legal entity groups a set of Application that can access Kernel Platform APIs.

A consent is required to gain access to non-automatic purposes.

Consents are managed directly by final users.

To know more about consents and how the Kernel Platform operates with them, refer to the consents documentation.

APIs

APIs are defined in the Kernel Platform through an OpenAPI 2.0 specification, also known as Swagger. This specification defines the endpoints that need to be exposed in the API Gateway along with the scopes that need to be created in the Authorization Server.

The only thing you need to add a new API to the Kernel Platform is a URL that serves the registration swagger.

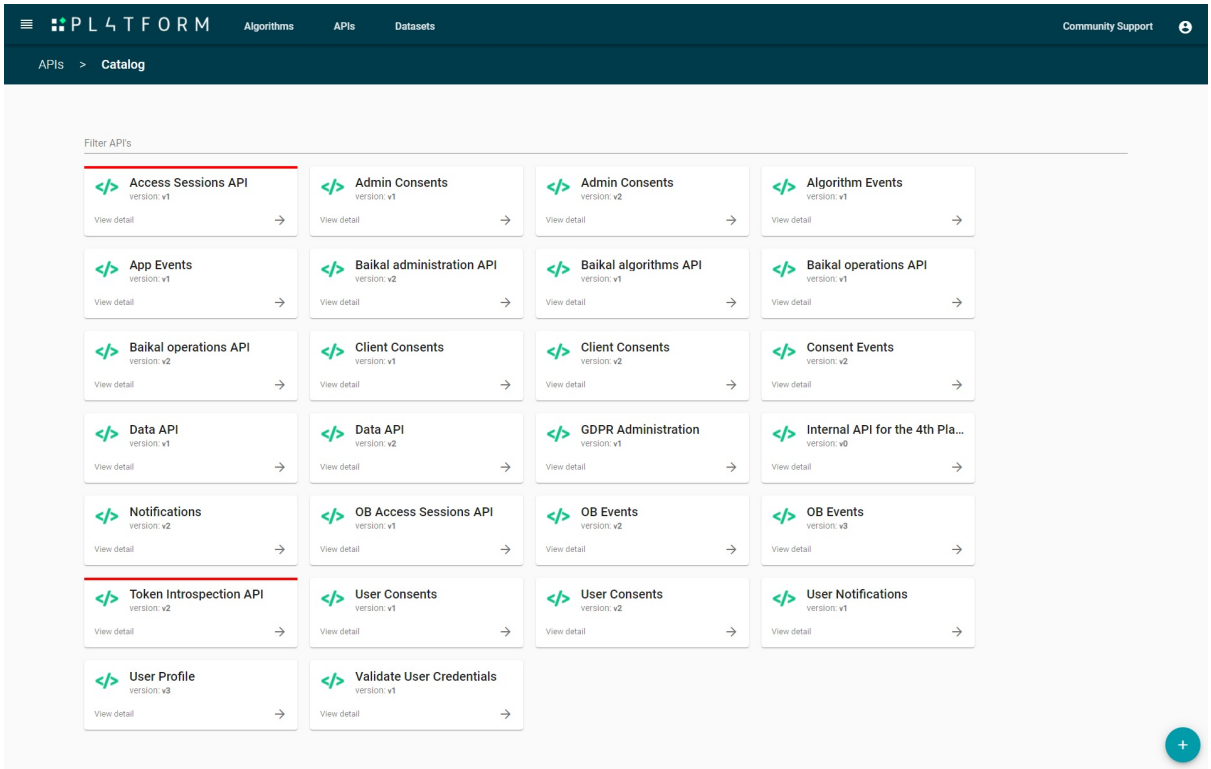
Checklist

Things to check BEFORE adding or updating an API:

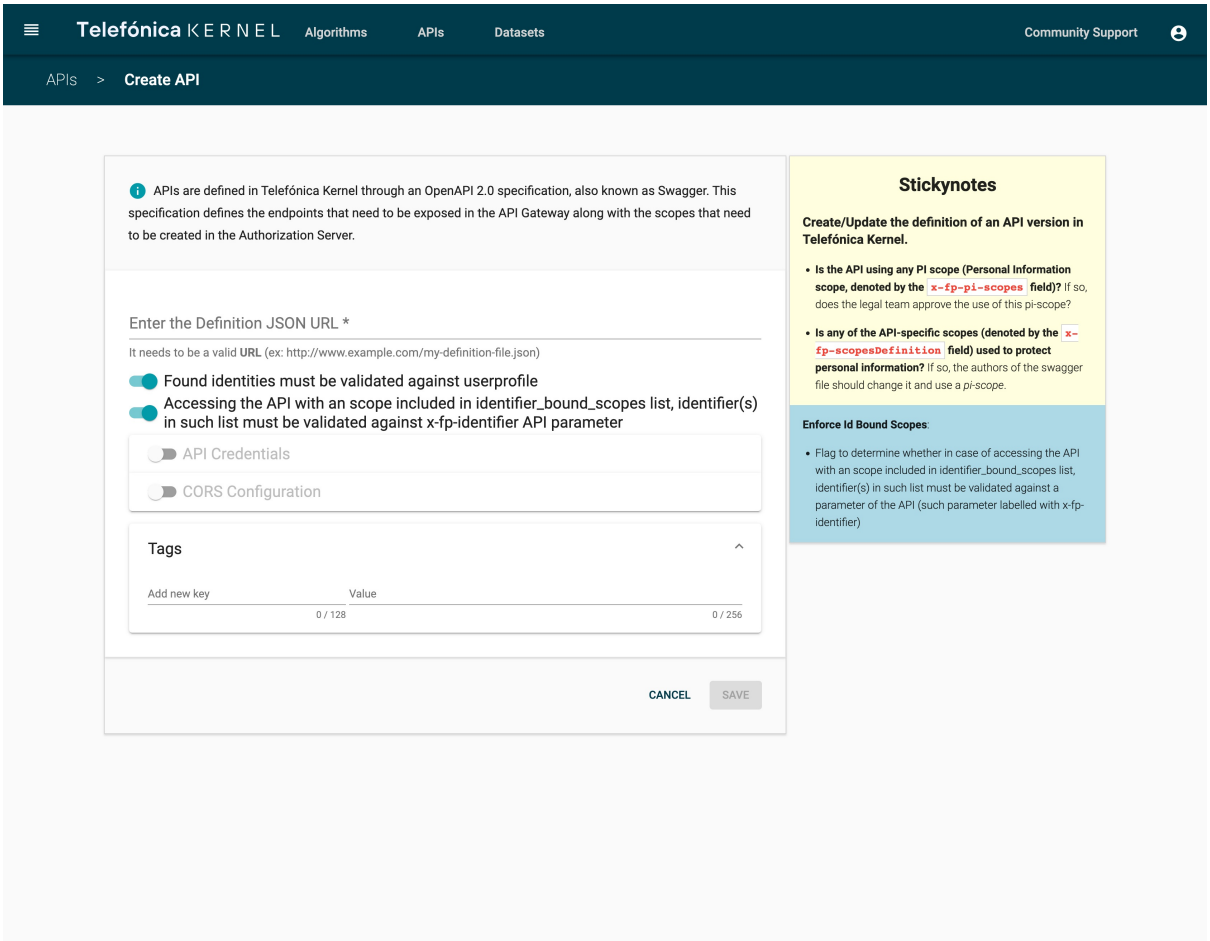
1. Is the API using any PI-Scopes (denoted by the `x-fp-pi-scopes` field)? If so, does the legal team approve the use of this PI-Scope and has it already been registered? Please go here to see how to register PI-Scopes.
2. Is any of the API-specific scopes (denoted by the `x-fp-scopesDefinition` field) used to protect personal information? If so, the authors of the swagger file should change it and use a PI-Scope.

Managing APIs

To get the list of already registered APIs, please go to the API sections and you will find the API catalog.

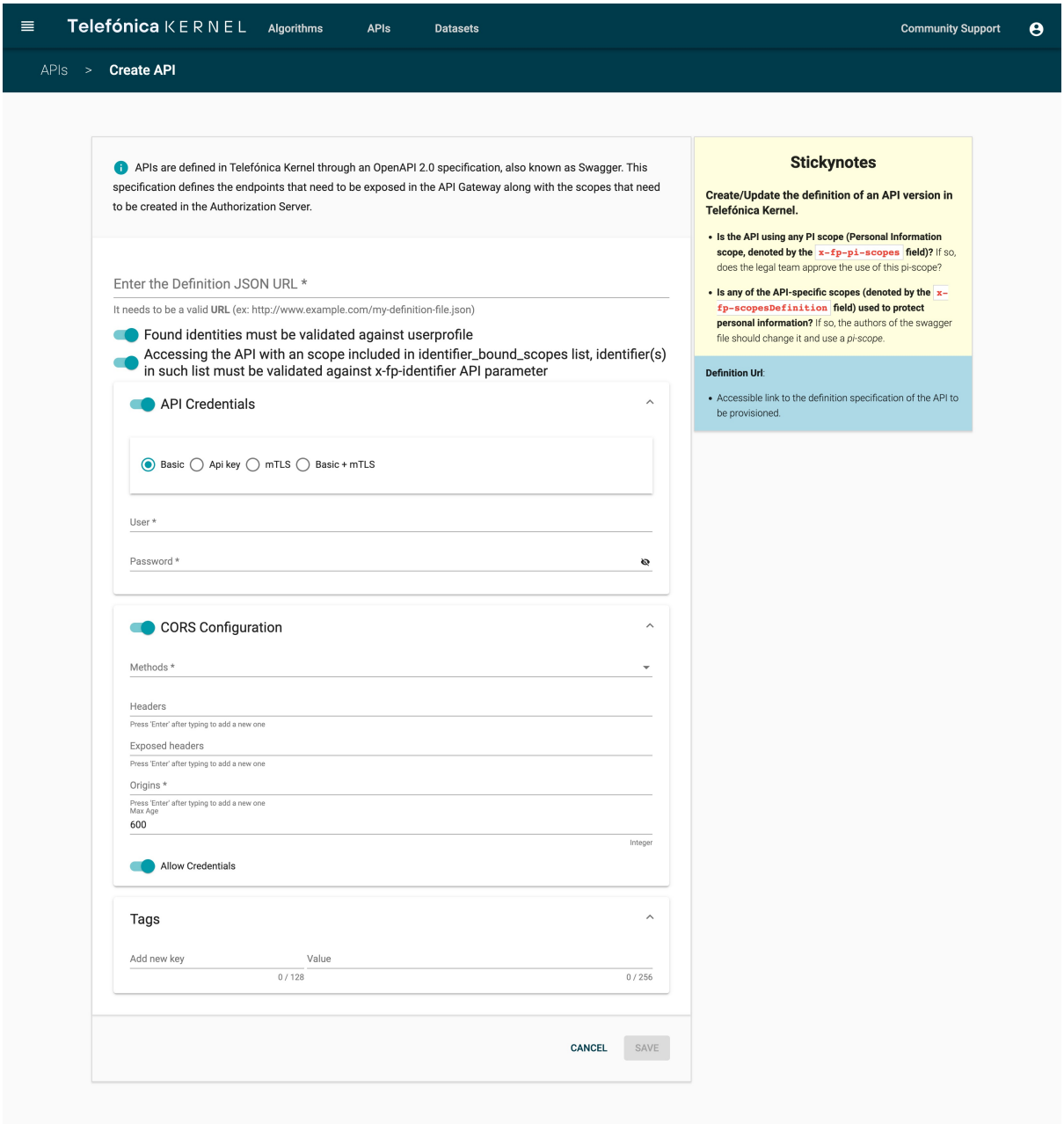


To create a new one, click on  at the bottom of the page and fill in a form like the following. In most of the cases you will just need to provide the public URL where the swagger is available.



- **Swagger URL**: URL where the swagger will be downloaded.
- **Found identities must be validated against userprofile**: flag to determine if the found identities must be validated against userprofile.
- **Tags**: list of labels that you want the Application to be associated with.

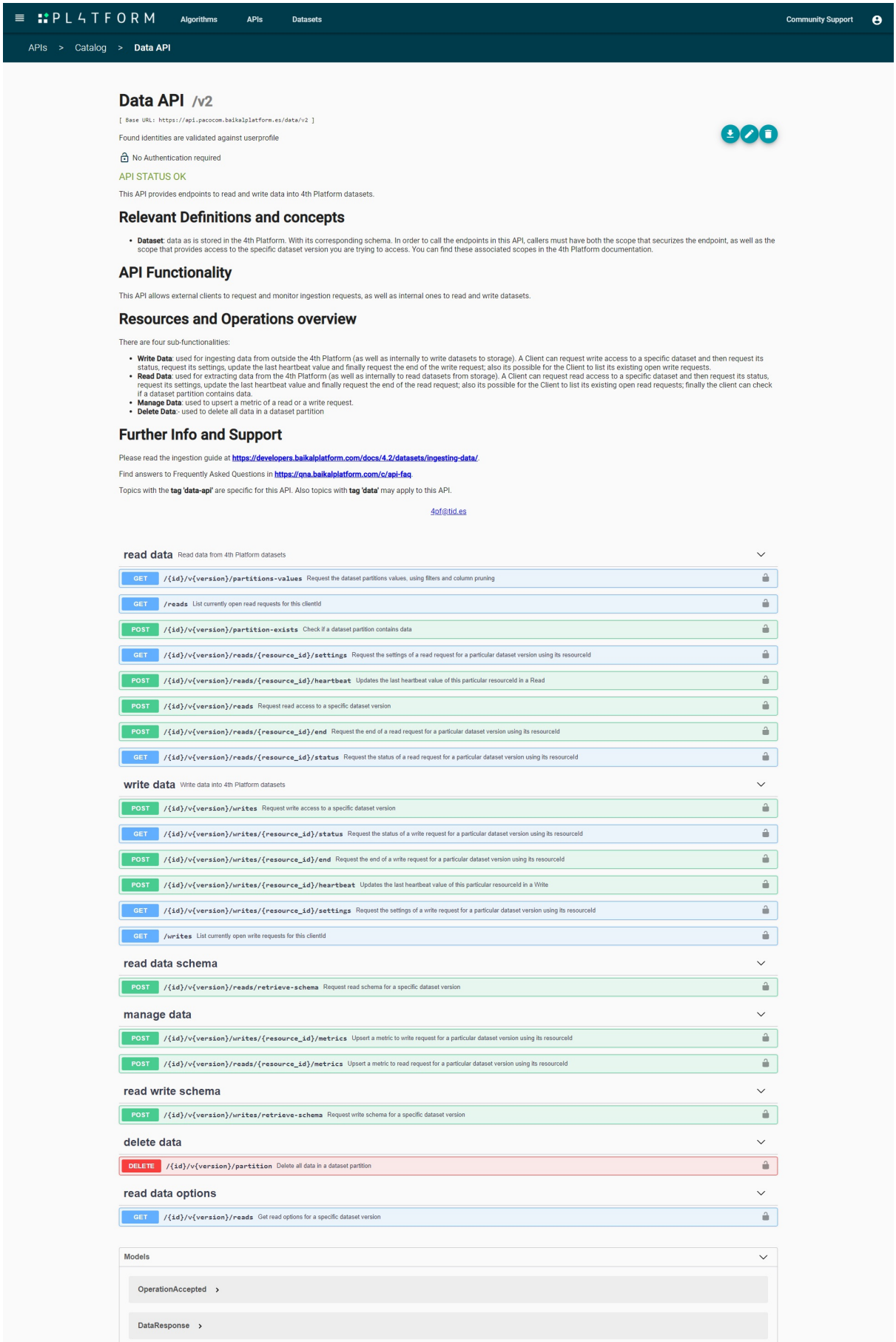
However, as reflected in the next image, you will need to provide further information in the following situations.



- when the API is directly exposed by an entity outside of the Kernel Platform or the OB (**API Credentials**), then you will need to provide the **credentials to access the backend endpoint**. There are four options, both considering the use of HTTPS towards the API backend:
  - configuring user and password for HTTP Basic Authentication.
  - configuring an API Key. In this case, you can specify the name of the HTTP header and the secret value that this header will have.
  - configuring mTLS authentication. In this case, you will need to provide client certificate and key and CA certificates so that both sides can authenticate.
  - configuring both Basic and mTLS authentication. Note that this option is only maintained for backward compatibility and it shall not be used.
- when the API is going to be accessed from JavaScript code served from domains different from the `baikalplatform.com` (**CORS Configurations**), then you will need to configure CORS, with the following options:
  - **Methods**: list of allowed HTTP verbs from the above domains code. This will determine the content of the `Access-Control-Allow-Methods` set by the Kernel Platform API server.

  - **Headers**: list of headers that are allowed to be included into HTTP requests to the API from the above domains code. This will determine the content of the `Access-Control-Allow-Headers` set by the Kernel Platform API server.
  - **Exposed Headers**: list of headers that JavaScript can read from responses. This will determine the content of the `Access-Control-Expose-Headers` set by the Kernel Platform API server.
  - **Origins**: list of trusted domains to serve JavaScript code that will have access to the API. This will determine the content of the `Access-Control-Allow-Origin` set by the Kernel Platform API server.
  - **Max age**: Indicated how long the results of the preflight request can be cached, in seconds.
  - **Allow credentials**: Flag to determine whether the Access-Control-Allow-Credentials header should be sent with true as the value.
- All APIs have a default rate limit of request per minute per JTI (token) configured in the environment. If you want change this limit for a client, you must configure the field rate_limit for that client. This configuration allows you to choose between 2 parameters (`jti` or `client_id`). JTI is preferred when you want to limit the number of requests that can be made with the same token. On the other hand, use `client_id` when you want to limit the traffic of an App globally. Also, you can set that limit for different time windows. First, you have to choose a mandatory value of requests per minute, but you can add others, such as requests per hour, month, and year if you want. For example, you may want that an app has a limit of 60 req/min to allow spikes of traffic but a max of 1200 req/hour (i.e. a mean value of 20 req/min).

After creating the API, you will see it appear along with the rest of registered APIs in the API catalog. If you click on the card of an API, you will see an API details page like the following, describing all the API endpoints and allowed operations. Note that using the buttons on the upper right corner, you can update or delete the API, as well as downloading the API swagger file that can be used by API clients to integrate with the Kernel Platform.



A swagger file with the API information can be download clicking on  .

An API can be deleted clicking on  .

An API can be edit clicking on  . Editing an API contains the same fields than creating an API.

Applications

An Application is an OAuth client that is able to consume APIs in the Kernel Platform.

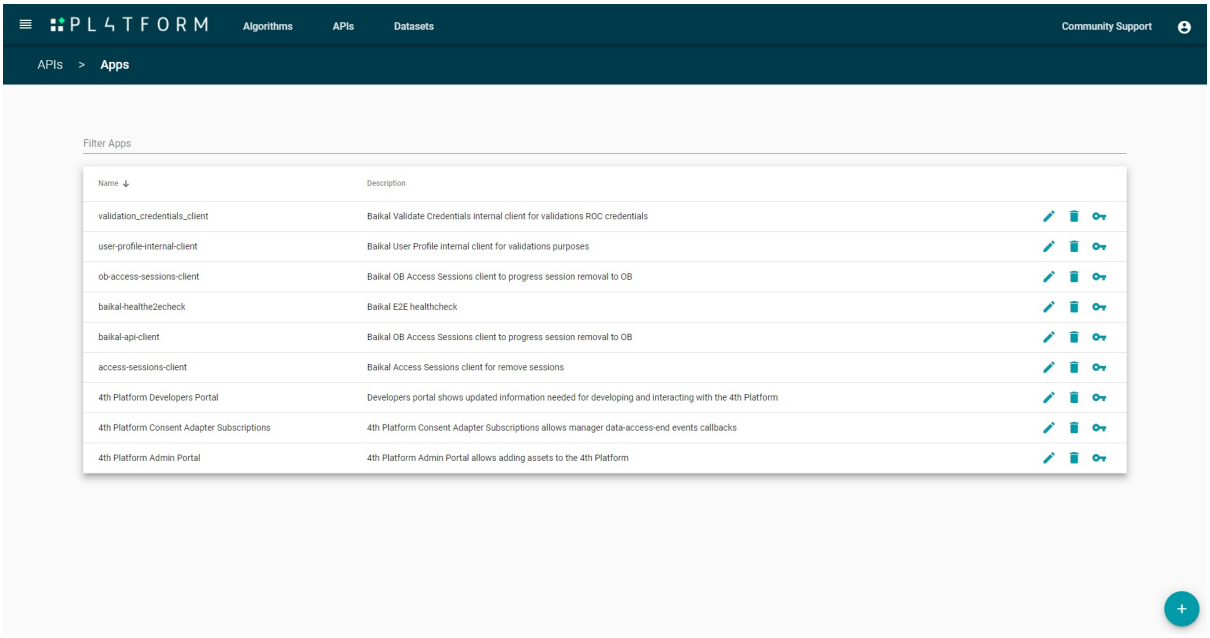
As explained in https://developers.baikalplatform.com/docs/latest/apis/getting-credentials/, Applications need to be registered into the Kernel Platform in order to get the required credentials to consume APIs. As an Administrator of the Kernel Platform you must control and validate this process as the mean to control what can be consumed by Applications and how.

Checklist

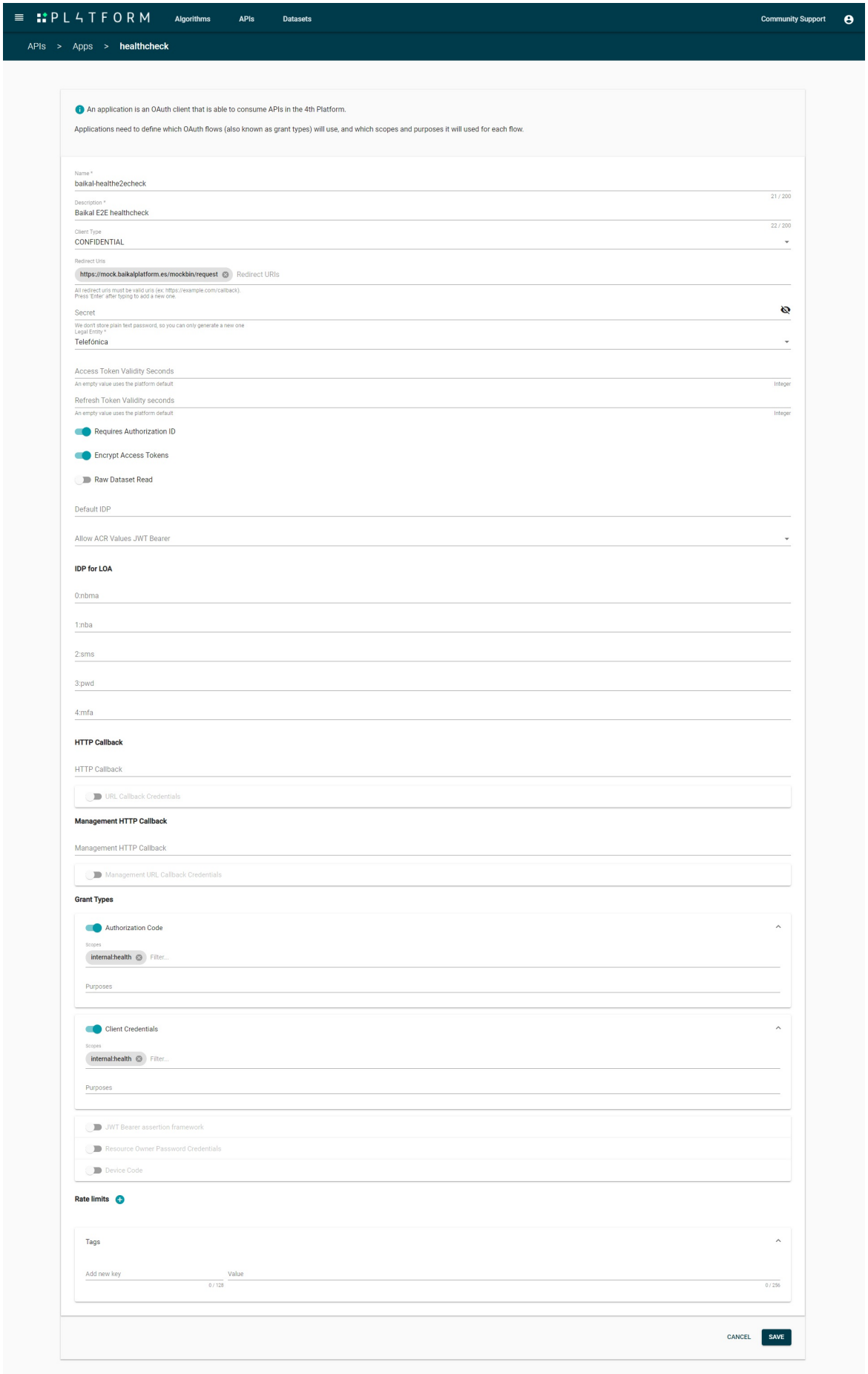
1. Is the Application requesting the minimal set of grant types, scopes and purposes it needs? If not, the unnecessary ones should be removed.
2. Is the Application requesting the JWT Assertion Framework grant-type with the `Requires Authorization ID` flag set to false? This is the equivalent to being a super-user when it comes to authorization: the Application will be able to impersonate any user when consuming scopes/PI-Scopes without the need to interact with the user. Few Applications should require this grant type. Make sure this is one of them before adding it to the platform.
3. Is the Application requesting the Client Credentials grant-type? This allows the Application to consume the scopes/PI-Scopes on its own name, without needing to do son within a user session. Few Applications should require this grant type. Make sure this is one of them before adding it to the platform.

Managing Applications

To get the list of already registered Apps, please go to the Applications sections and you will find list of APIs.

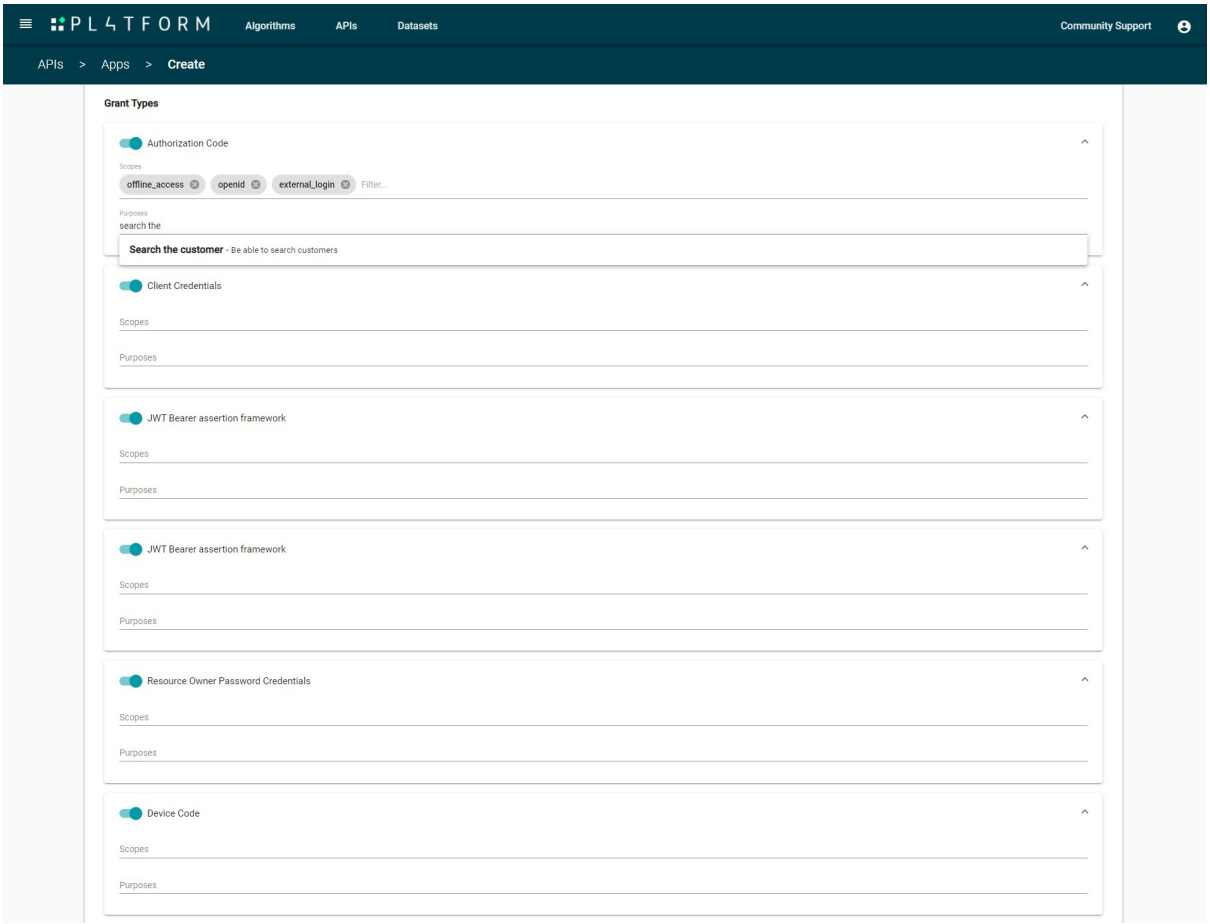


To create a new one, click on  at the bottom of the page and fill in a form like the following.



Where each fields has the following meaning:

- **Name**: short name of the Application.
- **Description**: description of the Application.
- **Id**: unique identifier of this Application. It is generated by the Kernel Platform.
- **Client Type**: the client type.
- **Redirect Uris**: the list of acceptable redirection URIs that can be used as callback.
- **Secret**: secret for the Application, make sure it is long and random enough.
- **Legal Entity**: legal entity the Application belongs to.
- **Access Token Validity Seconds**: define the expiration time of the access_token. If not included, Access Token expiration time will be the default one for the environment.
- **Refresh Token Validity Seconds**: define the expiration time of the refresh_token. If not included, Refresh Token expiration time will be the default one for the environment.
- **Requires Authorization Id**: whether this Application has to indicate a valid `authorization_id` when using the JWT Assertion framework to access APIs.
- **Encrypt Access Tokens** : defines if the access_token is encrypted or not.
- **Raw Dataset Read**: Defines if the app can access raw data (not pseudonymized).
- **Default IDP**: Default IDP for authenticate the clients of this app. In case this field is not set, 4P will use an IdP defined as default at platform level.
- **Allowed ACR Values JWT Bearer**: the list of acceptable ACR values for urn:ietf:params:oauth:grant-type:jwt-bearer.
- **IDP for LOA**: the default IDP for authenticate the clients of this app. Visit https://developers.baikalplatform.com/docs/4.3/apis/api-access/login.html#supported-acr-and-amr-values.
- **HTTP Callback**: URL where notifications will be sent by default for this Application. The Application can override this value when subscribing to notifications. This is an optional param that is only required for Apps that are actually expected to receive notifications.
- **Management HTTP Callback**: URL where management notifications will be sent by default for this Application. This notifications inform the Application about changes in user's consents that may (de)activate notification sending from the Kernel Platform to this Application and user. This is an optional param that is only required for Apps that are actually expected to receive notifications.
- **Grant Types**: this section activates different ways or grant-types the Application can use to get an access_token and thus gain access to the Kernel Platform APIs. Take a look to next image and the details after it to understand how to fill in this section.
- **Rate Limits**: the different Rate Limit that your app will support. If your client doesn't send this field, they will be had the default config of each environment.
- **Tags**: list of labels that you want the Application to be associated with.



Note the accessible scopes and purposes (and associated PI-Scopes) are configured individually for each grant-type, thus allowing to control what can be accessed depending on the method used to gain that access. Supported grant-types are as follows:

- **Authorization Code**: access is gained by following a regular login flow where the end-user authenticates to gain access to the APIs. The Application accesses the APIs in the name of the user. If you check the `offline access` flag then the Application will be able to refresh the access session without end user intervention, which is useful to avoid the user having to login continually every time she wants to use the Application. Additionally, this grant-type requires that a list of callbacks is registered in order to complete the login flow. You can see all details about this flow here.
- **Client Credentials**: access is done in the name of the Application itself, without any end-user interaction. You can see all details about this flow here.
- **JWT Bearer Assertion Framework**: access is done in the name of an end-user, without any end-user interaction. Note however that if the inclusion of a valid authorization_id is required, then the Application must have been contacted in the context of valid end-user session. You can see all details about this flow here.
- **Resource Owner Password Credentials**: the Resource Owner Password Credentials grant type.
- **Device Code**: the device Code grant type.

⚠ To gain a deeper knowledge on the different access types, please refer to the API access documentation.

⚠ Note that for using the JWT Assertion Framework, the Application will need to use a valid issuer. If the Application required a new to be set up, please refer to managing issuers.

After creating the Application, you will see it appears along with the rest of registered Applications in the Application section. There you can remove the Application or if you click on the Application, you will see an Application details page that is equivalent to the Application creation page, with the exception that the Client ID can't be changed and the Client secret can only be changed, but not queried. The rest of the fields can be edited to update the Application registration.

An app can be edit clicking on  . Editing an app contains the same fields than creating an app.

An app can be deleted cliking on  .

An app can be rotate cliking on  . This encryption key is used internally by the Kernel Platform to pseudonymize personal information for datasets this app has access to.

Roles

⚠️ Role-based access control is an optional feature that can be activated depending on business needs. Please, validate if the functionality is being used in your Kernel Platform. If it is not, then this section does not apply.

⚠️ When role-based access control is activated in the Kernel Platform the following configuration applies to the API access for ALL registered Applications and grant-types, therefore it is important to make sure it complies with business and security needs.

The Kernel Platform supports role-based access control to the APIs. This means that the Scopes and PI-Scopes that are authorized for each access session do not only depend on the the Application and the grant-type being used, but they also depend on:

- who the user is: applicable roles depend on the identifier (e.g. phone number) used to authenticate, as defined in the UserProfile response for that user.
- how the user authenticated: each role needs to meet a minimum security level (a.k.a. **LoA - Level of Authentication**) in order to be activated. You can check the list of defined LoA here.

Moreover, the Scopes and PI-Scopes can be granted in three different forms:

- **user-bound**: the access is granted for all the user identifiers (e.g. phone number within the user contract)
- **identifier-bound**: the access is granted only for the identifier (e.g. phone number) used for the authentication
- **role-bound**: the access is granted for the identifier (e.g. phone numbers) that share the same role within the UserProfile of that user.

All the above is configured within a roles table. Let's see it with an example. Imagine the next roles tables is configured in the Kernel Platform and the user logs in with an Application that is allowed to access any of the considered Scopes and PI-Scopes. Note Applications get at maximum the list of Scopes and PI-Scopes they are listed to get as per Application registration. Role-based access control will filter out Scopes and PI-Scopes from that static list.

Role	minum LoA	user-bound scopes	identifier-bound scopes	role-bound scopes
owner	3	`invoicing-read`, `mobile-quota-read`, `subscribed-products-user-read`, `user-consents-read`		
admin	2	`mobile-quota-read`, `subscribed-products-user-read`		`identifier-consents-read`
user	2		`mobile-quota-read`, `subscribed-products-phone-number-read`, `identifier-consents-read`	

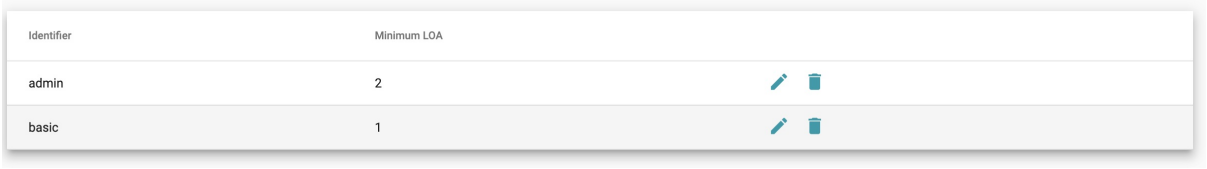
- If a user authenticates with LoA 3 (username and password), then it get access to `invoicing-read`, `mobile-quota-read`, `subscribed-products-user-read` and `user-consents-read`, for all user's lines and any other identifier.
- If a user authenticates with LoA 2 (SMS OTP) from an admin line, then the user gets access to `mobile-quota-read`, `subscribed-products-user-read` for all user identifiers and `identifier-consents-read` for all admin lines.
- If a user authenticates with LoA 2 (SMS OTP) from a user line, then the user gets access to `mobile-quota-read`, `subscribed-products-phone-number-read` and `identifier-consents-read` only for the phone number used to authenticate.

Checklist

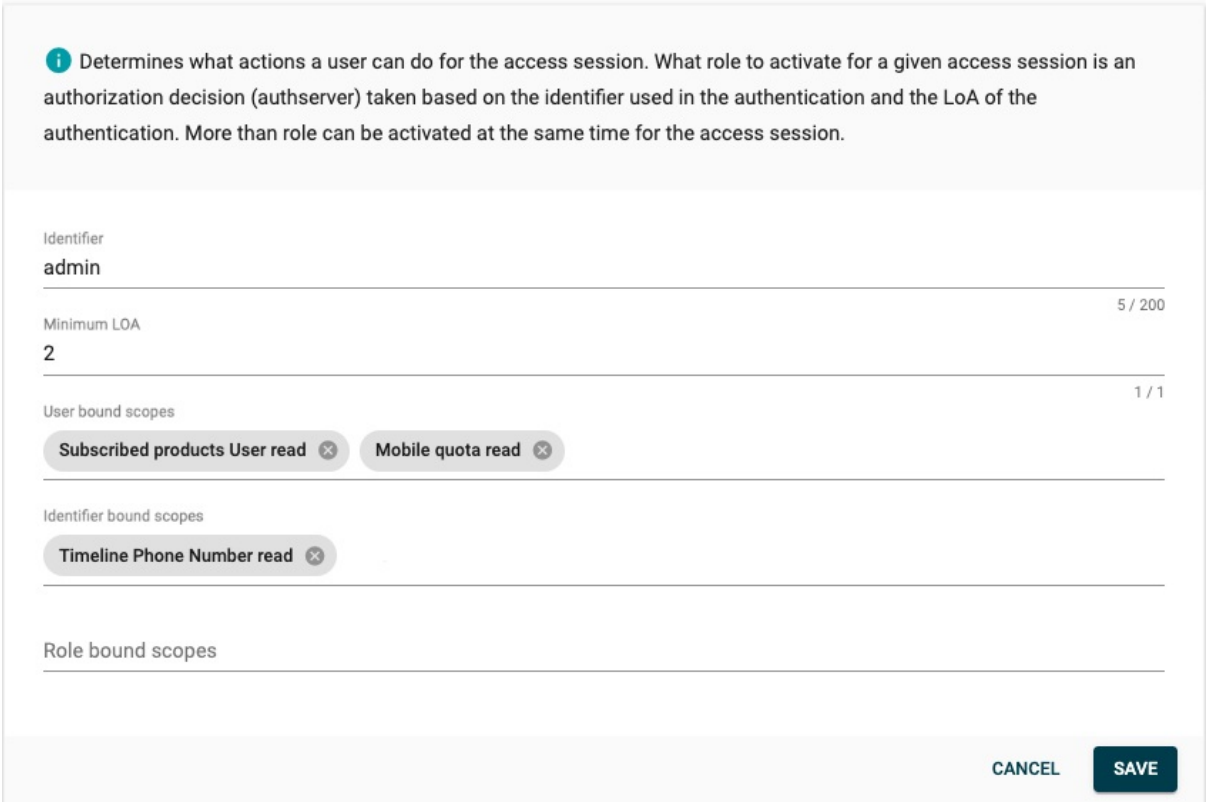
- It is important to make sure this configuration complies with your OB's business and security needs.
- Check the list of LoA that are active for the Kernel Platform deployment you are administering.

Managing roles

To get the list of already registered roles, please go to the Roles sections and you will find something like the following.



To create a new one, click the create button at the bottom of the page and fill in a form like the following.



- **Identifier**: name of the role
- **Minimum LoA**: minimum Level of Authentication that can activate this role
- List of Scopes and PI-Scopes that can be granted when this role is activated, in the three forms explained above in this page.

After creating the role, you will see it appears along with the rest of registered roles in the Roles section. There you can remove the role or update it if you click on the edit control.

Issuers

An issuer is a trusted entity that is able to create assertions used in OAuth's JWT Assertion Framework.

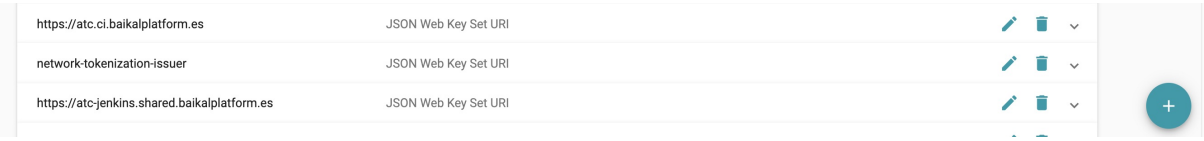
Checklist

Very few issuers should be created in the Kernel Platform. Most applications in the Kernel Platform should be able to work with the most popular OAuth flows (authorization code and client credentials).

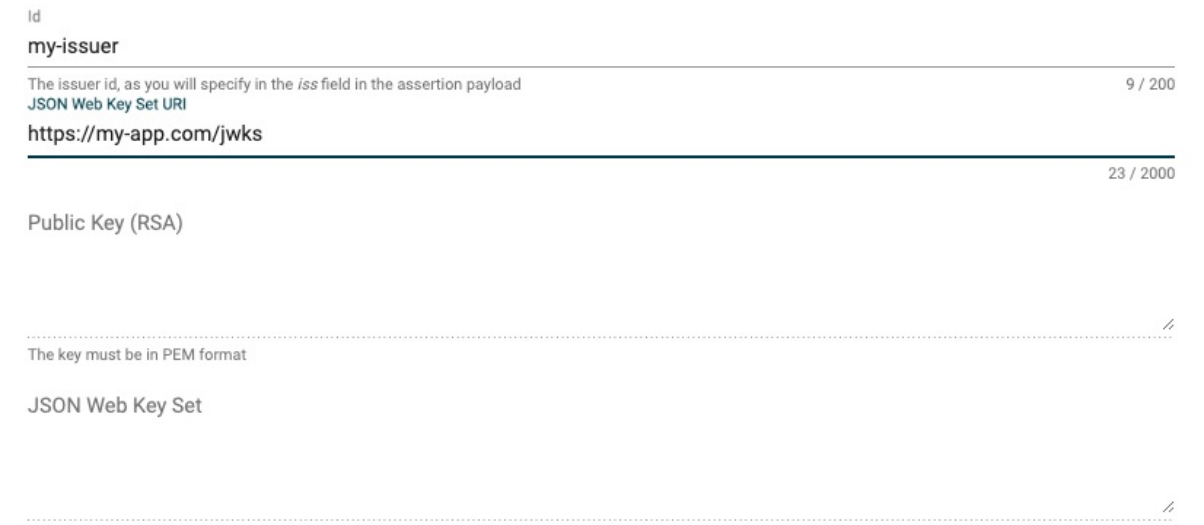
Before adding a new issuer, the requester should make a strong case as to why a new issuer is needed and why the existing issuers can't be reused.

Managing issuers

To get the list of already registered issuers, please go to the Issuers sections and you will find something like the following.



To create a new one, click the create button at the bottom of the page and fill in a form like the following.



In most of the cases you will just need to provide the public JWKS (JSON Web Key Set) URI. In other cases, you will to add the Public Key in PEM format corresponding to the Private Key of the issuer. Finally, you can also specify the JWKS content in JSON format directly, which will be something similar to the following:

```
{
  "keys": [
    {
      "kty": "RSA",
      "e": "AQAB",
      "use": "sig",
      "kid": "a8a87f21b1d3a11058ca4c6979c47a0640b8e437",
      "alg": "RS256",
      "n": "yWDCG4RYHZxQeCMgwubJSrtO6DHwq5JoEX5khfBrq7zEk1G3cg_8TOdtcRYp5xQpHYfjxAbdoYHeshvmMIxiF18mHFlKMRuLaI6
```

```
NuV_H81dqlapbGz_8HKsEH-gTq8Q_eeIKvMby_DbEJi2PfHboUDEmsqDx1HuqZApFNxmY8NZv90i4Eqm2s_PqpXWWzT7rT9GOLKwnY-HaPwfUII
bNA8Te7UchF7TLbaN1ptIY2WxTexYBsu0b615EU9WAxwR4tx_QktnOHWPkGBkNu7gWSD_F4AGXy_qfSxs0mcwZFpSxX6ZOLaICzkC7wjnFaqQDd
JXpmora4SlN_5AK9RAmWw"
    },
    {
      "kty": "RSA",
      "e": "AQAB",
      "use": "sig",
      "kid": "47d579ed5fb7fb2e7abf1865b94da1590ec2ce15",
      "alg": "RS256",
      "n": "nqF07XLzyiLMw76eGtU_7Qf8TELaiMisZzac1ueaSBo8Ew3E5LRCI5E_5B72DppwDKW1sL5o6QtzmyqUVYbbdZdWd3hP0ri-2-t
zURBhRGIdhKQMinnFP3FfU_RfLiI71v32YkDG-vHs9zeiFSbNHUVp7gPhtjJs84K7ExtPi0AYGCpmctflZTti7opz6yoTT-qHrx3HAB84J9KySD
S8lBxDSokQpkgLHY9uCUXR3x049ArRGPmCuKwKP8WYv5_lDaPECZ13LtTwJAbLx-249Sca7m06e6u9RnuXyzRBQViDn6DbrjVluxfxo_NeO8vQx
-zefNzwjf8kEqdiFBmDIw"
    }
  ]
}
```


After creating the issuer, you will see it appears along with the rest of registered issuers in the Issuers section. There you can remove the issuer or update it if you click on the edit control.

Big Data

The Kernel Platform provides the functionalities to read and write huge amounts of information that is made available to Algorithms to process them, producing as result new information. Information is organized in different sets that we simply refer to as Datasets.

The URM is a Telefónica standard to normalize Datasets throughout the company. You can list and get details on URM Datasets for the Kernel Platform here.

Datasets

Each Dataset is associated with a specific domain (e.g. information on mobile lines in the OB, users location, call CDRs, etc) and complies with a predefined format. Registering a Dataset implies two different steps:

- Firstly, the Dataset must be registered and a description of the nature of the information it holds must be provided. Additionally, as an Administrator you will be able to choose the storage type:
  - Batch: information is stored in the cloud object storage capability.
  - Low-latency: information is stored in a database. This type has been deprecated already from Iro/4.1 release onwards, so we HIGHLY RECOMMEND NOT TO USE IT.
- Secondly, a version of the Dataset needs to be registered. In this step, the specific schema of the information is provided, in the form of an Avro schema following the Avro 4P specifications.

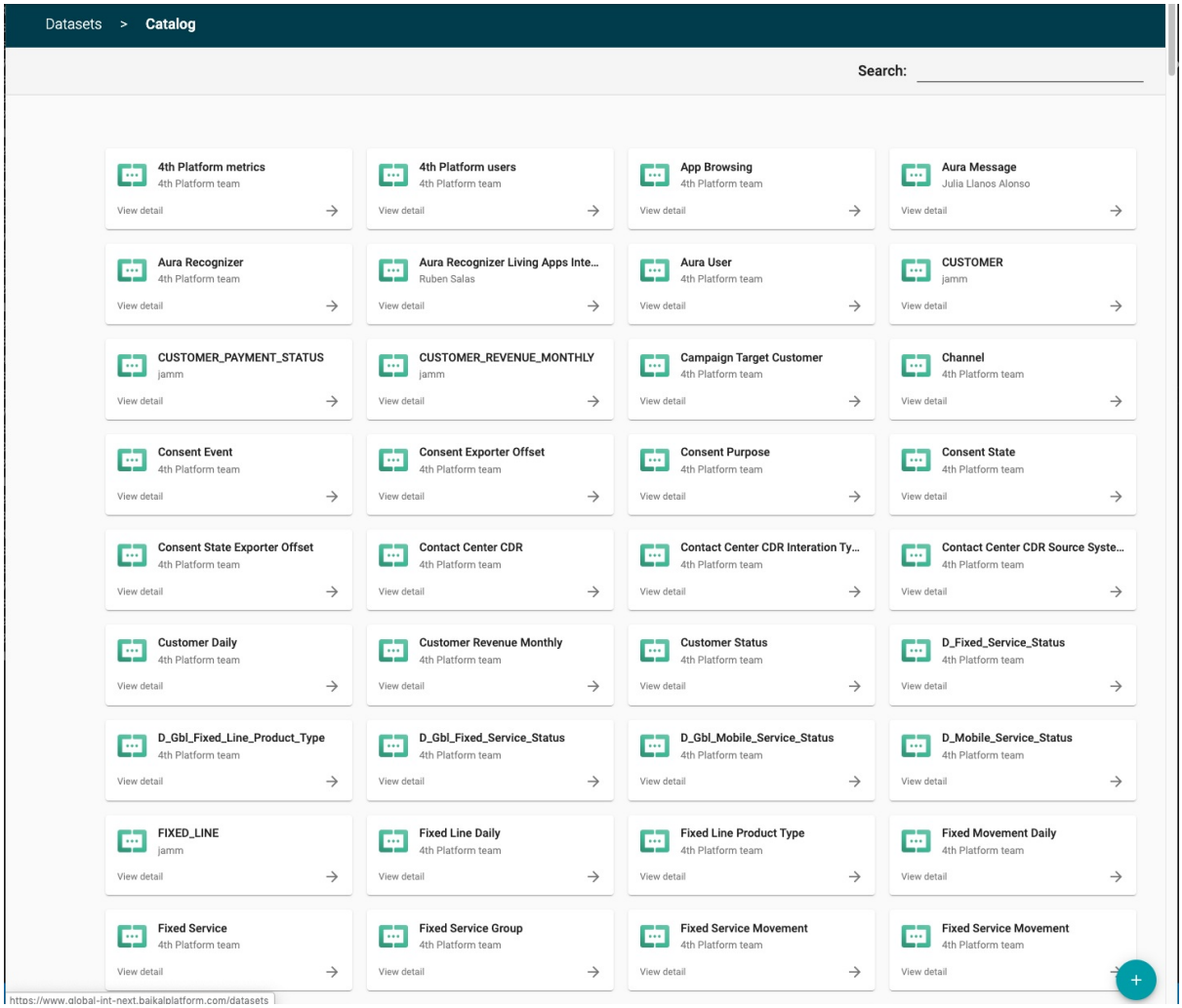
The next sections explain how to perform both steps. If you want to learn further details on the internal operation please refer to here

Checklist

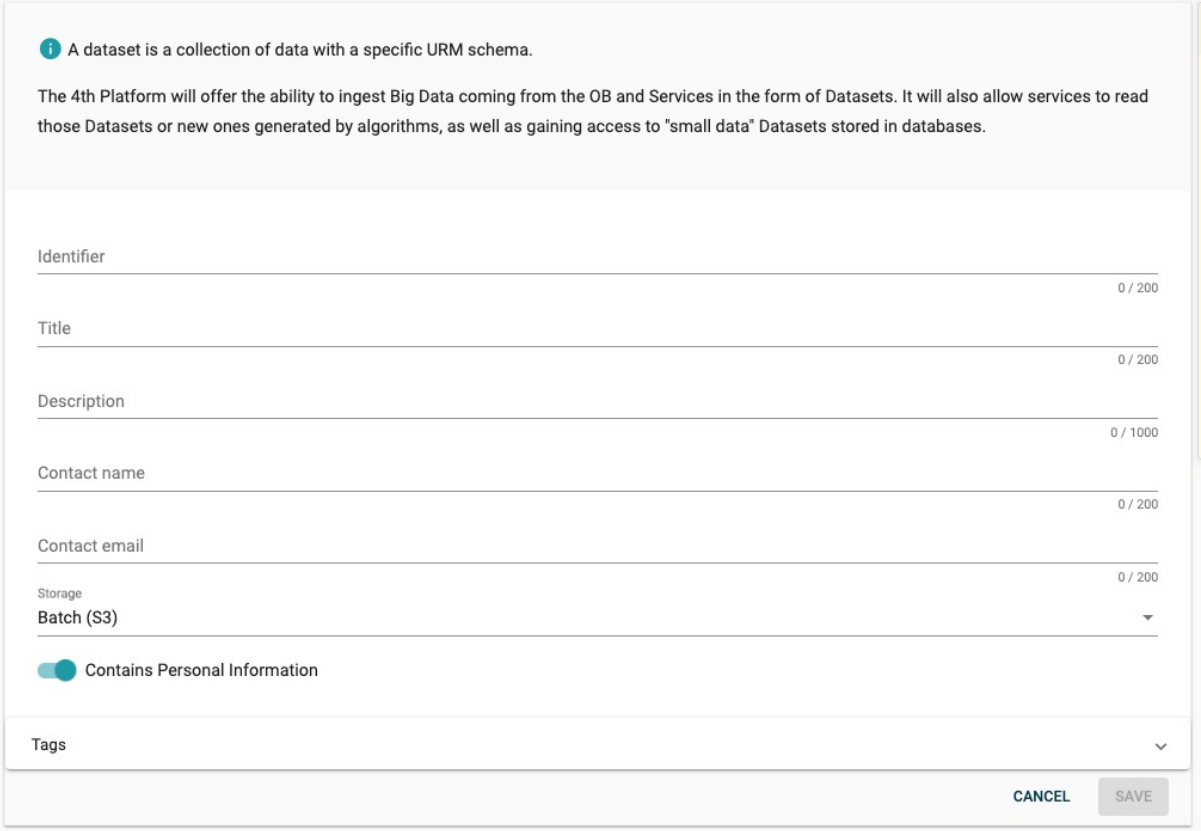
1. Make sure if the Dataset holds Personal Information and if so reflect it when registering it.
2. Make sure the provided schema complies with Avro 4P specs. All Dataset schemas that you can get from 4P Datasets are compliant and are also part of the URM standard.
3. Have a clear understanding on how data needs to be partitioned so that information processing can the scale correctly to big amount of information.
4. In case the Dataset holds personal information, create the read and write PI-Scopes BEFORE creating the Dataset version.

Managing Datasets

To get the list of already registered Datasets, please go to the Dataset sections and you will find the Dataset catalog.

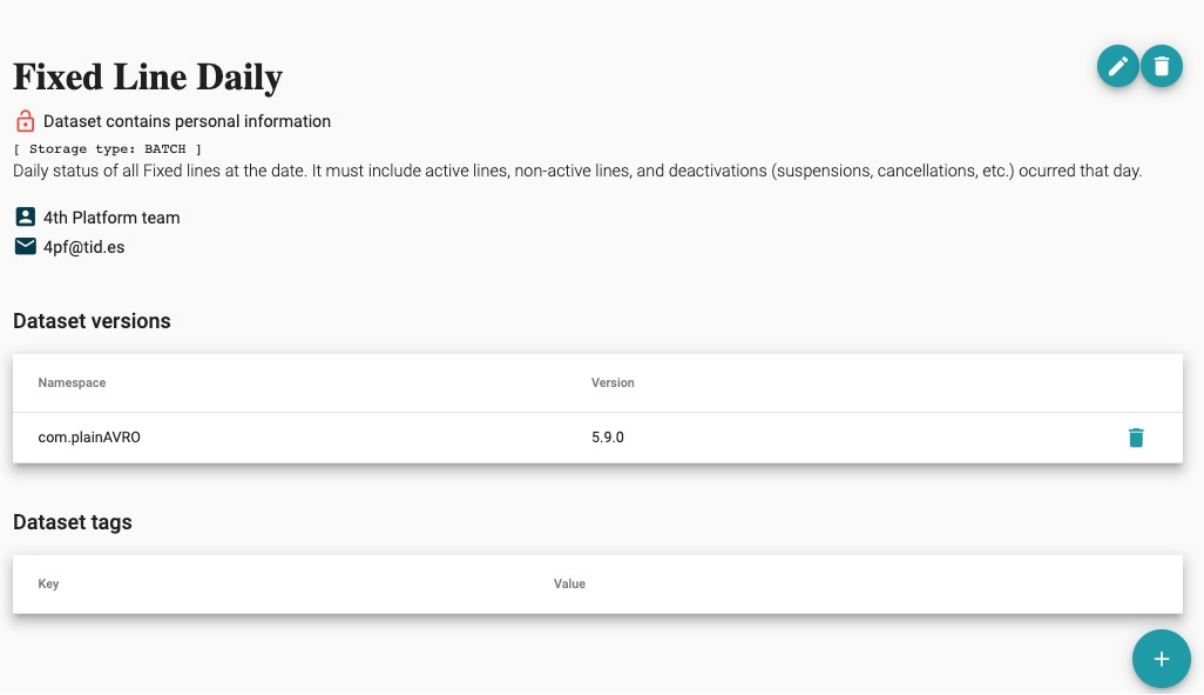


To create a new one, click the create button at the bottom of the page and fill in a form like the following. Note that for official URM Datasets, you can get almost all the required information from here.



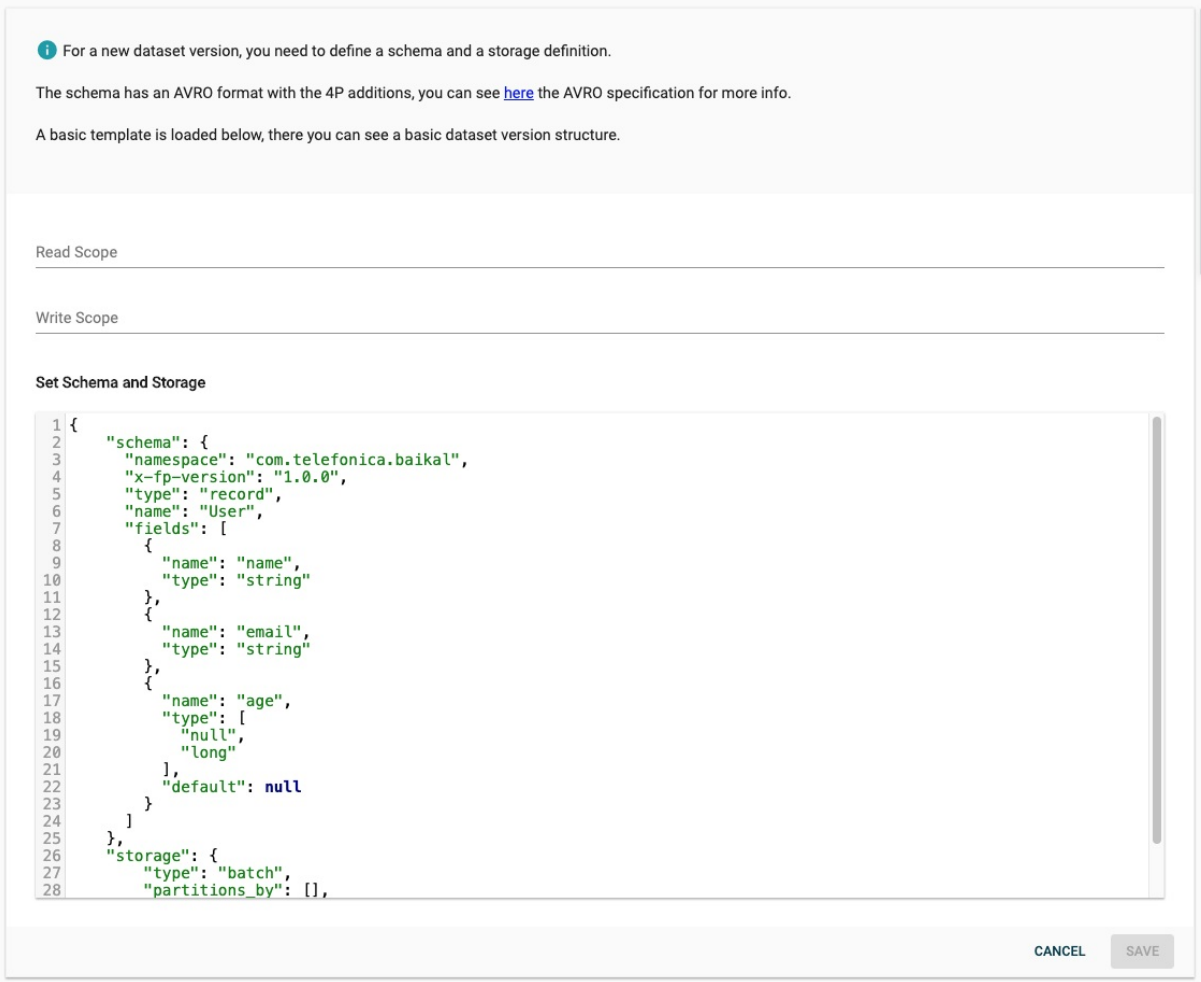
- **Identifier**: short name to identify the Dataset.
- **Title**: short description that is intended to be understood by humans.
- **Description**: longer description explaining the nature and meaning of the information hold by the Dataset
- **Contact name** and **Contact email**: contact details for the source of this information.
- **Storage**: type of storage. Use only `Batch`.
- **Contains Personal Information**: boolean flag to signal if this Dataset holds Personal Information.
- **Tags**: key and value pairs to freely attach metadata that can be used to search, organize, etc this Dataset within the overall catalog.

After creating the Dataset, you will see it appears along with the rest of registered Datasets in the Dataset catalog. If you click on the card of a Dataset, you will see a Datast details page like the following Note that using the buttons on the upper right corner, you can update or delete the Dataset definition.



⚠️ Datasets are versioned with three digits `x.y.z`. `x` is the major version number. Changes to `y` or `z` are minor upgrades that have to be backwards and forward compatible. Note the version is reflected in the value of the `x-fp-version` param within the Avro schema of the Dataset.

On the Dataset details page, you can add a Dataset version by clicking on the create button, through a form like the following. Note that for official URM Datasets, you can get some of the required information (schema, storage type, etc) from here.



- **Schema and Storage**: json content including the Avro schema the Dataset needs to comply with, as well as the details on the storage type including two important pieces of information (for further details you can refer to here):
  - how to **partition** data through the `partition_by` field. This property is used to partition data inside the object storage using different folders. It improves performance for data accesses when filters are used inside the algorithms and enables partial updates based on partition key values.
  - how to **order** data through the `sort_by` field. This property is used to write the data sorted by specific key values inside the storage files. It improves performance for data accesses when filters are used inside the algorithms and helps avoiding additional sort operations in some algorithms.

In case the Dataset holds personal information, you will need to specify the PI-Scopes that allow to read and write this Dataset. Remember you have to create them BEFORE being able to register the Dataset version. After that, type the PI-Scope in the corresponding fields of the form and select the correct one from the drop-down menu.

- **Read Scope**: This is the PI-Scope that will allow to read the Dataset.
- **Write Scope:**: This is the PI-Scope that will allow to write the Dataset.

⚠ For different Dataset versions you can set different PIScopes. This is not expected to be the common situation but can happen to cope wit cases where a new version adds new Personal Information that deserves a different treatment.

⚠ In order to distinguish Dataset scopes from APIs scopes, a RECOMMENDED would practice is to prefix the Datasets scopes with the string `data-`, for example, `data-my-awesome-dataset-read`.

After creating the Dataset version, you will see it appears along with the rest of registered versions in the Dataset details page. You can request the deletion of Dataset version, or, if you click on one version, you will see a page with the same form that you used to create the version. You can update the Dataset schema, the storage type or the scopes there if you need to.

⚠️ Note that deleting a Dataset version deletes ALL data actually being stored by the Kernel Platform in the object storage, so data is lost.

Algorithms

The Kernel Platform provides an environment for the execution of Big Data Algorithms using Spark. As a simplified explanation, Algorithms read Datasets, compute information, and then write Datasets as the result of that computation, and this can be repeated multiple times until the final result is achieved.

In a similar way as it is done for APIs, the Kernel Platform controls that Algorithms access information within the Datasets only for those users that they actually can. This is achieved via Purposes and PI-Scopes. Users grant o revoke access to the PI-Scopes listed in the Purposes associated to each Algorithm.

For further information about platform algorithms, please refer to this section.

Checklist

- Make sure the legal department has agreed on the purposes that the Algorithm needs to be associated with, before allowing it to be registered into the Kernel Platform.

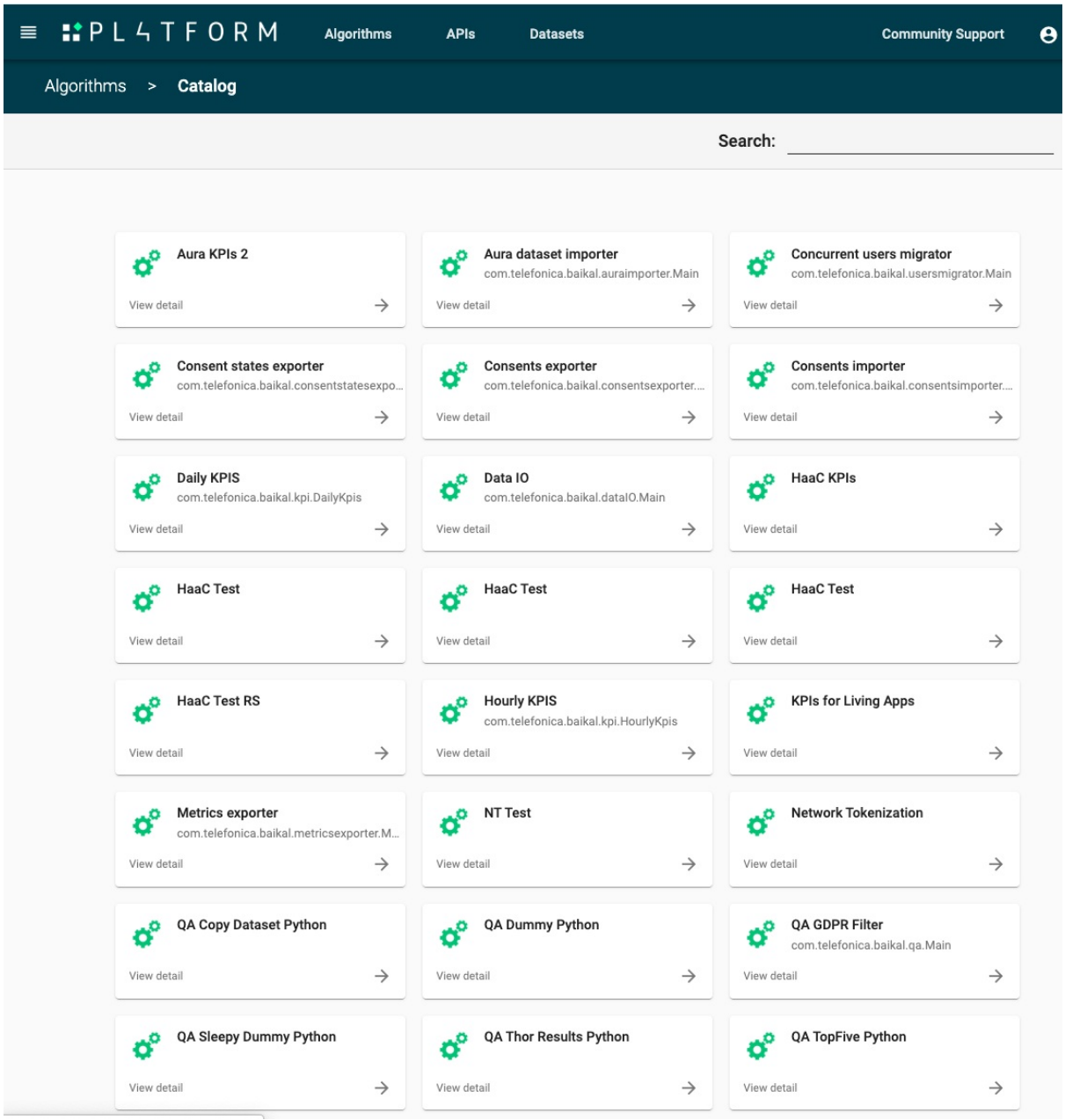
Managing Algorithms

Currently the Admin Portal does not allow to register Algorithms. To do so, the procedures explained in Kernel Platform Guides and Kernel Platform QnA must be followed.

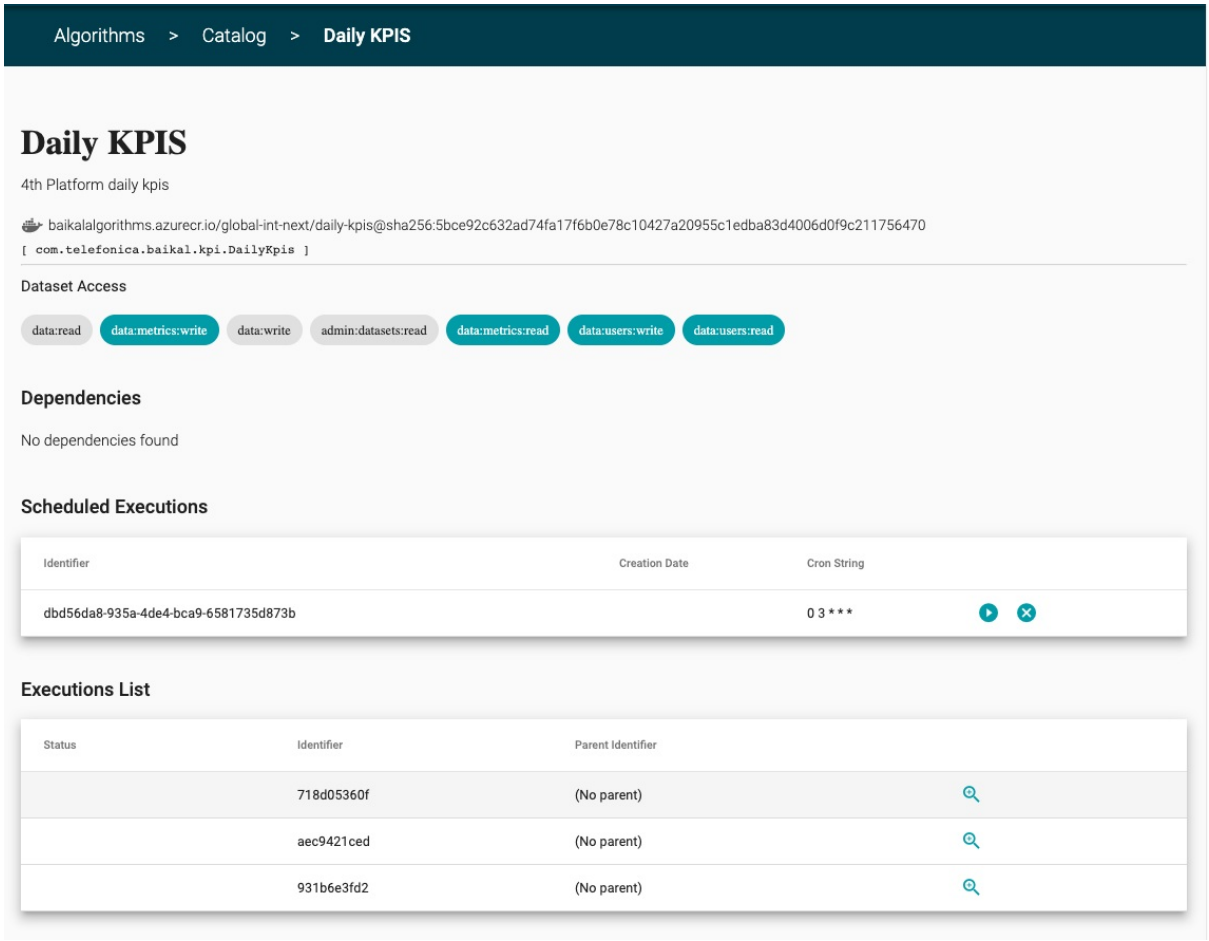
Once the Algorithm is registered, the Admin Portal does allow to manage it, in different ways:

- getting information on registered Algorithms
- running them on demand or regularly
- getting access to execution history and logs

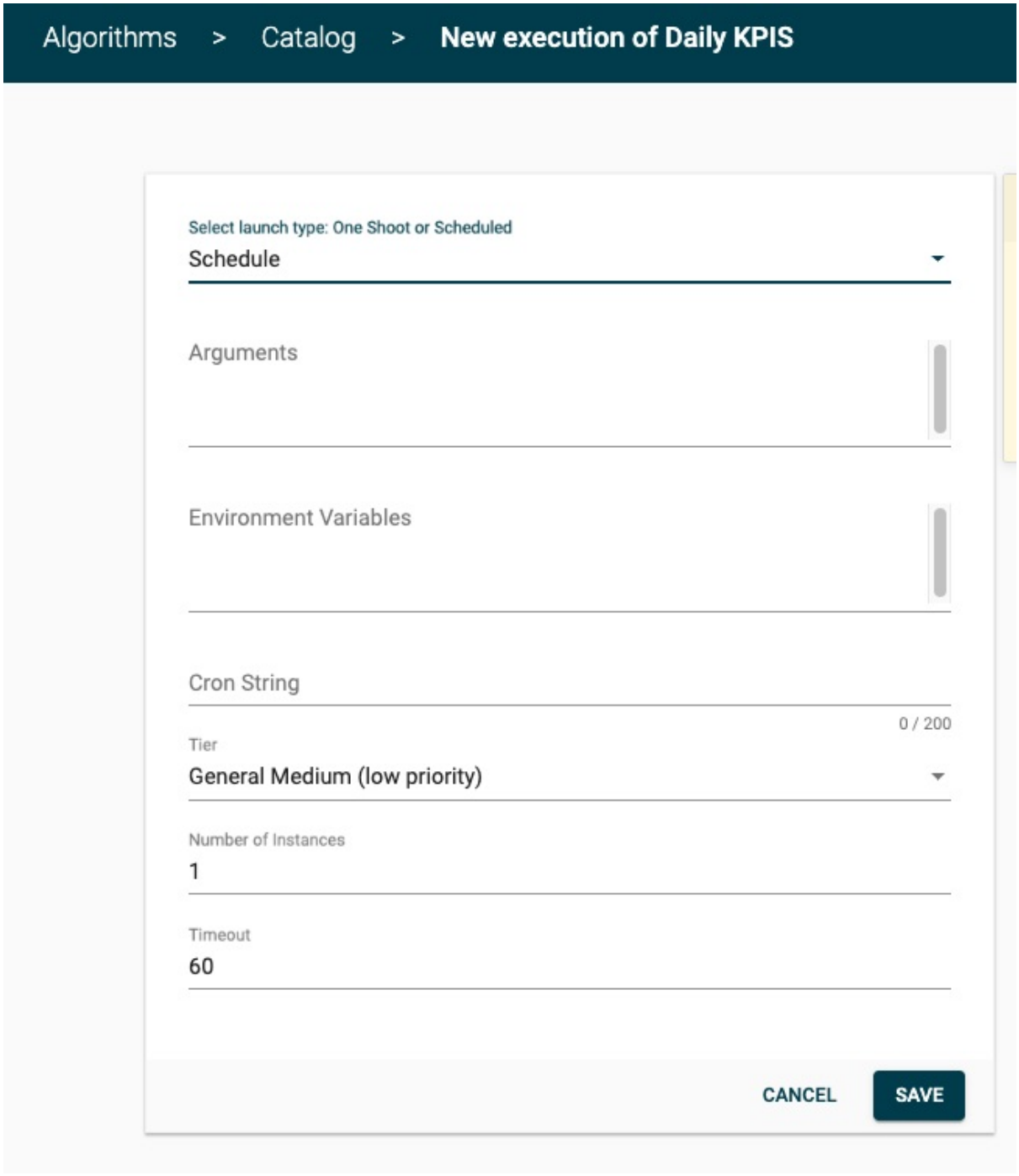
To get the list of already registered Algorithms, please go to the Algorithms sections and you will find the Algorithms catalog.



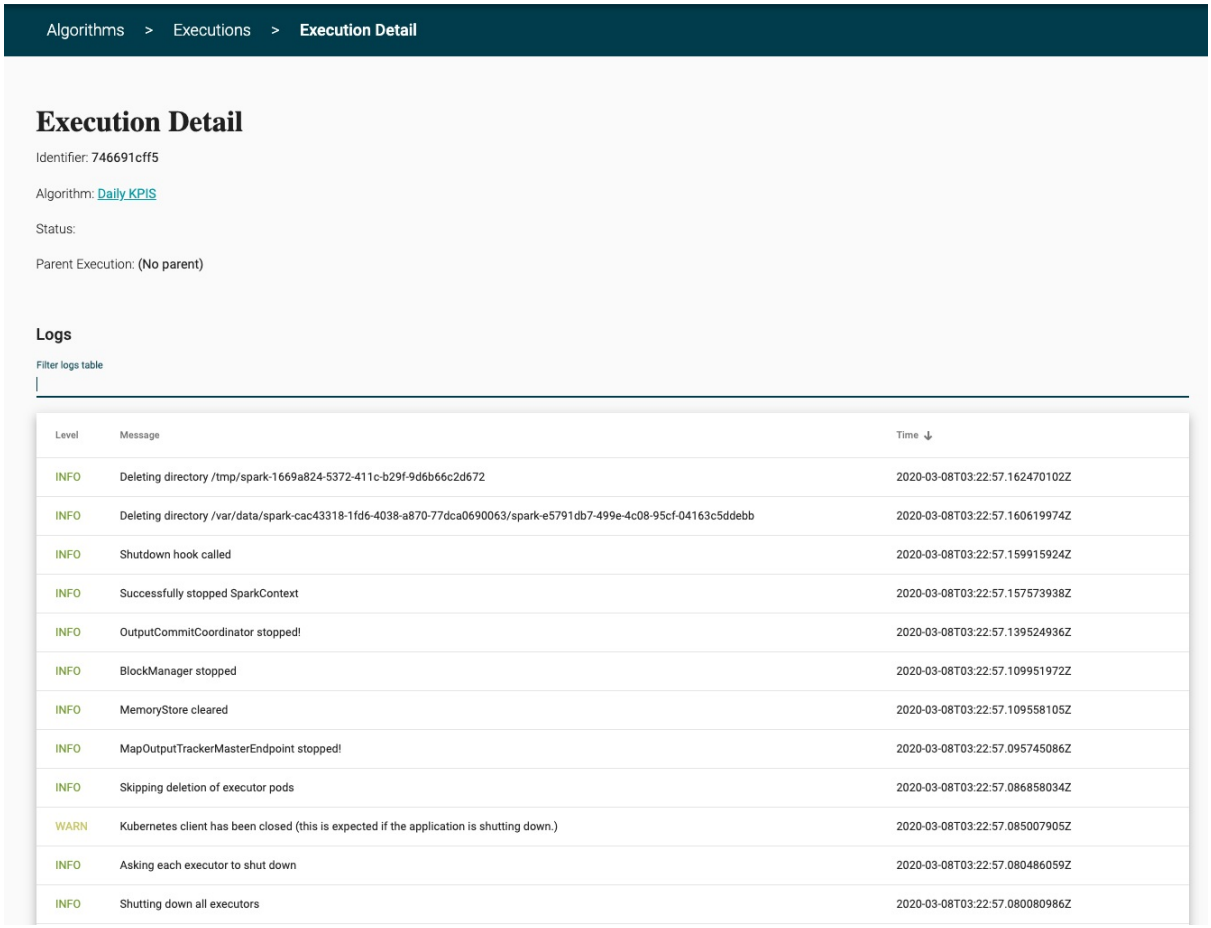
By clicking on one Algorithm you will see a page like the following. There you will see the list of Scopes and PI-Scopes to access Datasets, as well as information on regularly scheduled executions and the list of past executions for that Algorithm.



By clicking on the create button, at the lower-right corner, you will be able to create new executions of this Algorithm by filling a form like the following. Please refer to here for all the details on how to fill in the different parameters.



If you click on a past execution you will then see the execution details, where you will get access to execution logs.



Lastly, as shown in the next image, note that you can get the list of executions for all the algorithms registered in the platform by going to the "Executions" section within the left menu. By clicking on a specific execution you will see the same information as reflected in the above image.