Solution Details Guide

This document aims to explain:

- Each of the components required to deploy MongoDB on Kubernetes.
- Each of the components required to deploy Serverless function to SAP BTP, Kyma Runtime

Open DMC_NextNumber_InAppExtensions/batch-nn-mongo-db/code_solution directory, it contains nine YAML files, known as "Kubernetes manifest". The manifest is a specification of a Kubernetes API object in JSON or YAML format. A manifest specifies the desired state of an object that Kubernetes will maintain when you apply the manifest.

Let's get started with the MongoDB setup.

What is MongoDB?

MongoDB is a document-oriented NoSQL database used for high-volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB uses collections and documents. Documents consist of key-value pairs, which are the basic unit of data in MongoDB. Collections contain sets of documents and functions equivalent to relational database tables.

Why do we use it for a current solution?

We need to store a new sequence value for the Batch number generated in Kyma serverless function. The goal was to use a database that doesn't require schema defined beforehand. Since MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents. This makes MongoDB very flexible and adaptable to real business world situations and requirements. Each document can be different, with a varying number of fields. The size and content of each document can be different from each other.

- Define MongoDB Secrets in mongodb-secrets.yaml

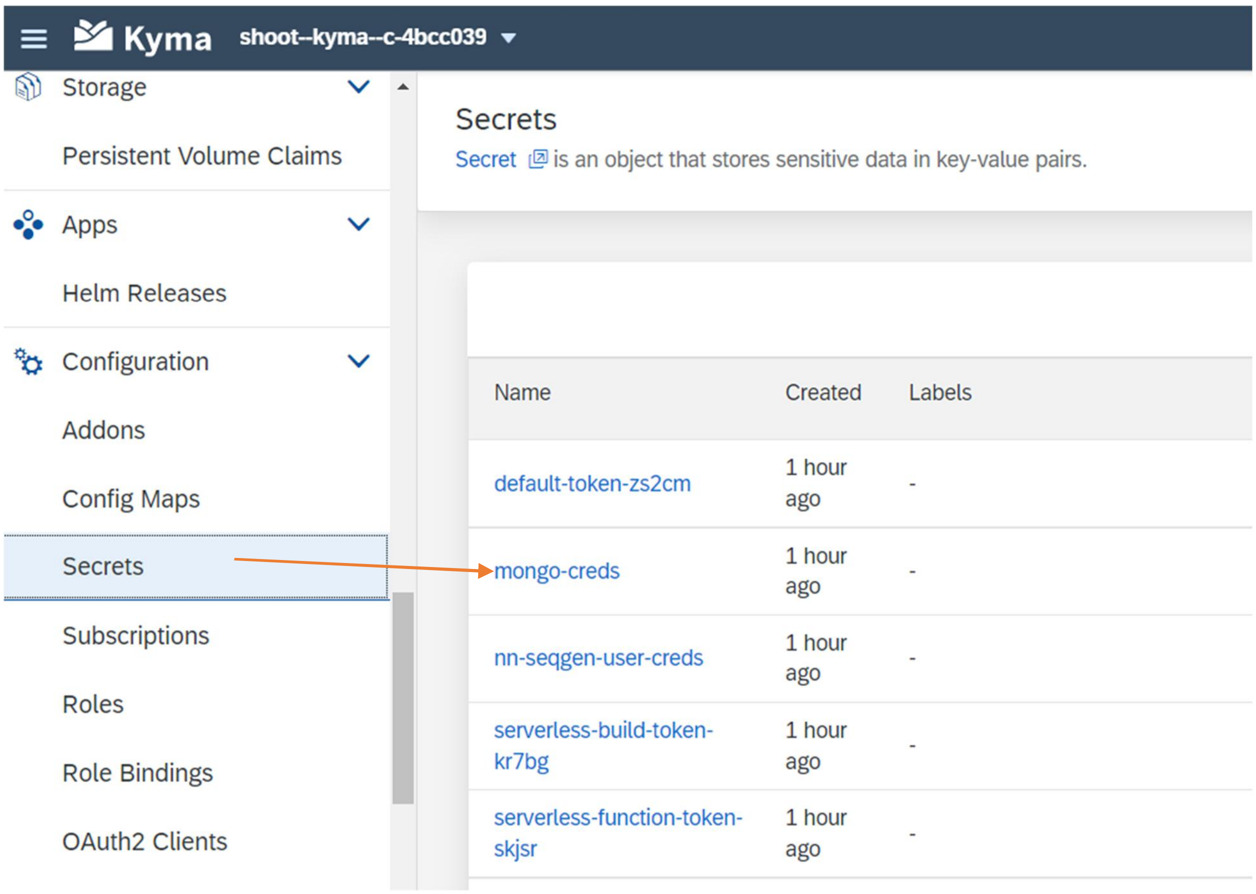
Secrets in Kubernetes are the objects used for supplying sensitive information to containers. For the security of our MongoDB instance, it is wise to restrict access to the database with a password. Secret is used to store confidential data in key-value pairs, in our case it is user name and password that will be used to login to the MongoDB instance.

Defines the user name and password which is base64-encoded.

```
apiVersion: v1
data:
  password: cGFzc3dvcmQxMjM= #password123
  username: YWRtaW51c2Vy #adminuser
kind: Secret
metadata:
  name: mongo-creds
```

Important Note: Kubernetes stores the content of all secrets in a base64-encoded format.

The result of this deployment can be found if you navigate to Configuration à Secrets in Kyma Console UI.



Secrets

Secret is an object that stores sensitive data in key-value pairs.

Name	Created	Labels
default-token-zs2cm	1 hour ago	-
mongo-creds	1 hour ago	-
nn-seqgen-user-creds	1 hour ago	-
serverless-build-token-kr7bg	1 hour ago	-
serverless-function-token-skjsr	1 hour ago	-

- Define MongoDB Persistent Volume in mongodb-pvc.yaml

We require volumes to store the persistent data. In this way, the data is not lost even if our pod goes down.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: mongo-data
spec:
 accessModes:
  - ReadWriteOnce
 resources:
  requests:
   storage: 1Gi
```

- Deploying the MongoDB in mongodb-deployment.yaml

We use the official mongo image from the docker hub to deploy MongoDB to Kubernetes.

```
apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
   app: mongo
 name: mongo
spec:
 replicas: 1
 selector:
   matchLabels:
     app: mongo
 strategy: {}
 template:
   metadata:
     labels:
       app: mongo
   spec:
     containers:
     - image: mongo
       name: mongo
       args: ["--dbpath","/data/db"]
       livenessProbe:
         exec:
           command:
             - mongo
             - --disableImplicitSessions
             - --eval
             - "db.adminCommand('ping')"
         initialDelaySeconds: 30
         periodSeconds: 10
         timeoutSeconds: 5
         successThreshold: 1
         failureThreshold: 6
       readinessProbe:
         exec:
           command:
             - mongo
             - --disableImplicitSessions
             - --eval
             - "db.adminCommand('ping')"
         initialDelaySeconds: 30
         periodSeconds: 10
         timeoutSeconds: 5
         successThreshold: 1
         failureThreshold: 6
       env:
       - name: MONGO_INITDB_ROOT_USERNAME
         valueFrom:
           secretKeyRef:
             name: mongo-creds
             key: username
       - name: MONGO_INITDB_ROOT_PASSWORD
         valueFrom:
           secretKeyRef:
```

```
          name: mongo-creds
          key: password
      volumeMounts:
      - name: "mongo-data-dir"
        mountPath: "/data/db"
    volumes:
    - name: "mongo-data-dir"
      persistentVolumeClaim:
        claimName: "mongo-data"
```

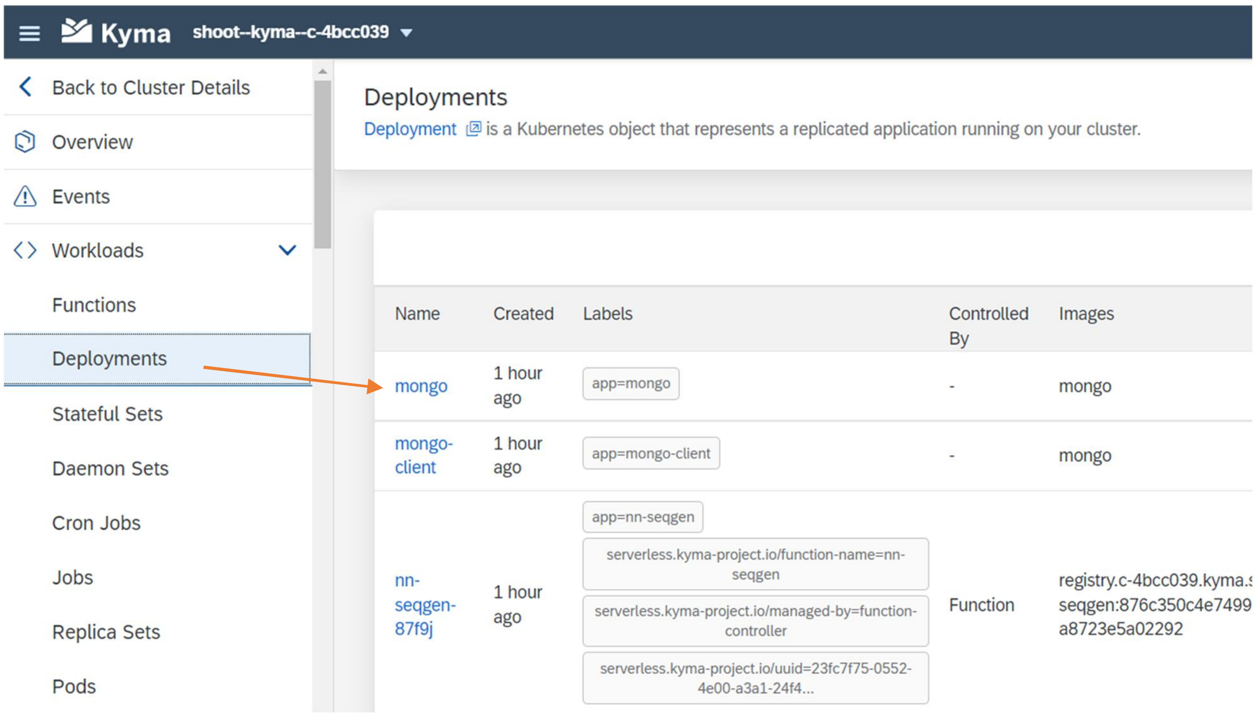
The deployment YAML of MongoDB has a lot of components such as env vars from secrets, probes.
Password to the MongoDB database must be supplied securely to the MongoDB container; that is why we injected env vars through the secrets.

```
      env:
      - name: MONGO_INITDB_ROOT_USERNAME
        valueFrom:
          secretKeyRef:
            name: mongo-creds
            key: username
      - name: MONGO_INITDB_ROOT_PASSWORD
        valueFrom:
          secretKeyRef:
            name: mongo-creds
            key: password
```

Probes: Probes ensure that the container does not get stuck in a loop due to any bug and can be restarted automatically if an unexpected error occurs.

```
      livenessProbe:
        exec:
          command:
            - mongo
            - --disableImplicitSessions
            - --eval
            - "db.adminCommand('ping')"
        initialDelaySeconds: 30
        periodSeconds: 10
        timeoutSeconds: 5
        successThreshold: 1
        failureThreshold: 6
      readinessProbe:
        exec:
          command:
            - mongo
            - --disableImplicitSessions
            - --eval
            - "db.adminCommand('ping')"
        initialDelaySeconds: 30
        periodSeconds: 10
        timeoutSeconds: 5
        successThreshold: 1
        failureThreshold: 6
```

The result of this deployment can be found if you navigate to Workloads à Deployments in Kyma Console UI.



Deployments

Deployment is a Kubernetes object that represents a replicated application running on your cluster.

Name	Created	Labels	Controlled By	Images
mongo	1 hour ago	app=mongo	-	mongo
mongo-client	1 hour ago	app=mongo-client	-	mongo
nn-seqgen-87f9j	1 hour ago	app=nn-seqgen serverless.kyma-project.io/function-name=nn-seqgen serverless.kyma-project.io/managed-by=function-controller serverless.kyma-project.io/uuid=23fc7f75-0552-4e00-a3a1-24f4...	Function	registry.c-4bcc039.kyma.s seqgen:876c350c4e7499 a8723e5a02292

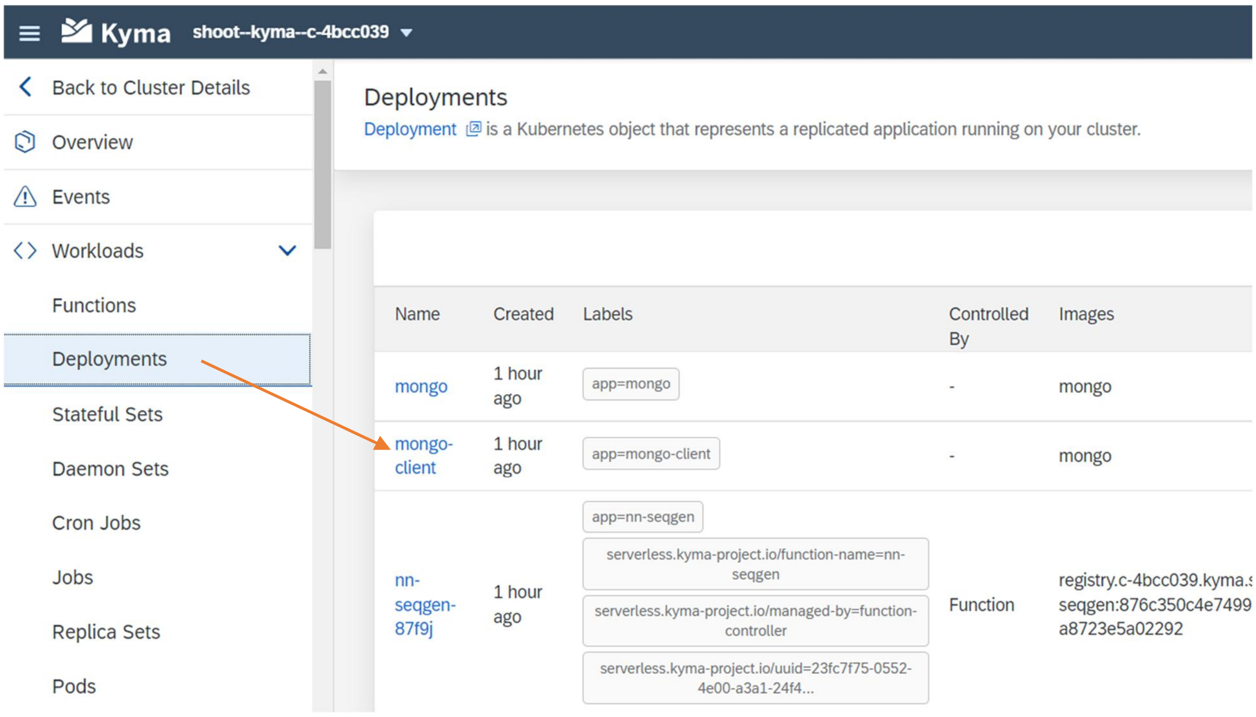
· Define Mongo client for exploring MongoDB shell in mongodb-client.yaml

After creating the MongoDB instance, we may wish to run basic commands inside it. Define a dedicated mongo client from where we will access the MongoDB database.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: mongo-client
  name: mongo-client
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongo-client
  template:
    metadata:
      labels:
```

```
      app: mongo-client
    spec:
      containers:
      - image: mongo
        name: mongo-client
```

The result of this deployment can be found if you navigate to Workloads à Deployments in Kyma Console UI.



Deployments

Deployment is a Kubernetes object that represents a replicated application running on your cluster.

Name	Created	Labels	Controlled By	Images
mongo	1 hour ago	app=mongo	-	mongo
mongo-client	1 hour ago	app=mongo-client	-	mongo
nn-seqgen-87f9j	1 hour ago	app=nn-seqgen serverless.kyma-project.io/function-name=nn-seqgen serverless.kyma-project.io/managed-by=function-controller serverless.kyma-project.io/uuid=23fc7f75-0552-4e00-a3a1-24f4...	Function	registry.c-4bcc039.kyma.s seqgen:876c350c4e7499 a8723e5a02292

After the client gets created, follow the below steps to access MongoDB.

Ø Use kubectl exec to open a bash command shell where you can execute commands:
`kubectl exec -n dev deployment/mongo-client -it -- /bin/bash`
The command should return similar to that: `root@mongo-client-69bfd49fdd-f65tj:/#`

Ø Login into the MongoDB shell:
`mongosh --host mongo-nodeport-svc --port 27017 -u adminuser -p password123`

The command should return:

```
Current Mongosh Log ID: 61cb050e3834a953a14edc4a
Connecting to:        mongodb://mongo-nodeport-svc:27017/?directConnection=true
Using MongoDB:        5.0.5
Using Mongosh:        1.1.6
```

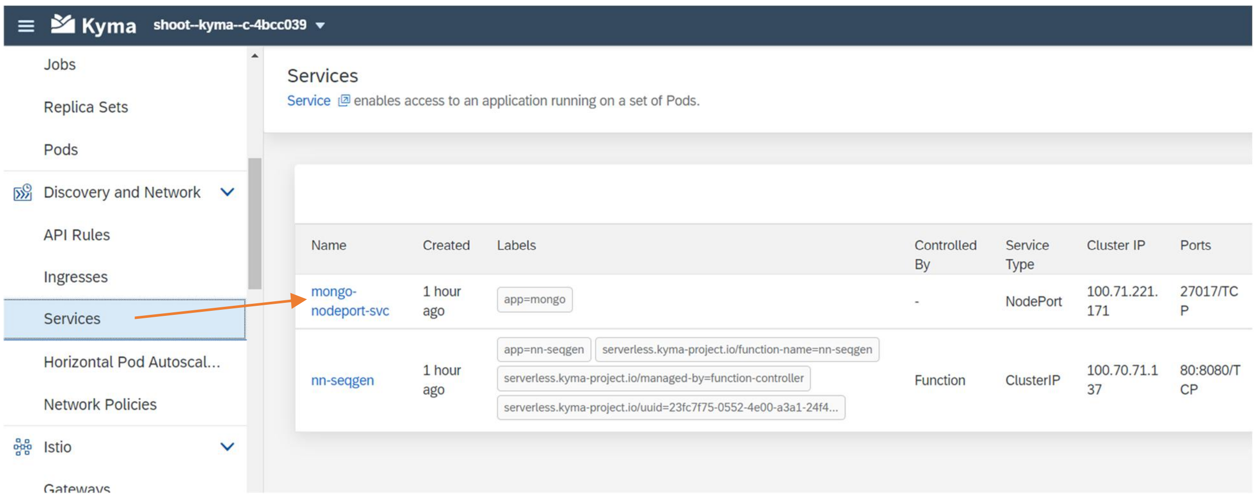
Important Note: log in to MongoDB shell with user name and password defined in mongodb-secrets.yaml

- Define Node Port type service in mongodb-nodeport-svc.yaml

This service is needed to connect to MongoDB from a serverless function.

```
apiVersion: v1
kind: Service
metadata:
 labels:
  app: mongo
 name: mongo-nodeport-svc
spec:
 ports:
 - port: 27017
  protocol: TCP
  targetPort: 27017
 selector:
  app: mongo
 type: NodePort
```

After service was deployed, we can connect to MongoDB using host: mongo-nodeport-svc and port:27017



Services

Service enables access to an application running on a set of Pods.

Name	Created	Labels	Controlled By	Service Type	Cluster IP	Ports
mongo-nodeport-svc	1 hour ago	app=mongo	-	NodePort	100.71.221.171	27017/TCP
nn-seqgen	1 hour ago	app=nn-seqgen serverless.kyma-project.io/function-name=nn-seqgen serverless.kyma-project.io/managed-by=function-controller serverless.kyma-project.io/uuid=23fc7f75-0552-4e00-a3a1-24f4...	Function	ClusterIP	100.70.71.137	80:8080/TCP

MongoDB is now set up and ready to use.

The following pages are about YAMLs provided to create a serverless function with API Rule, Secrets, and Config map.

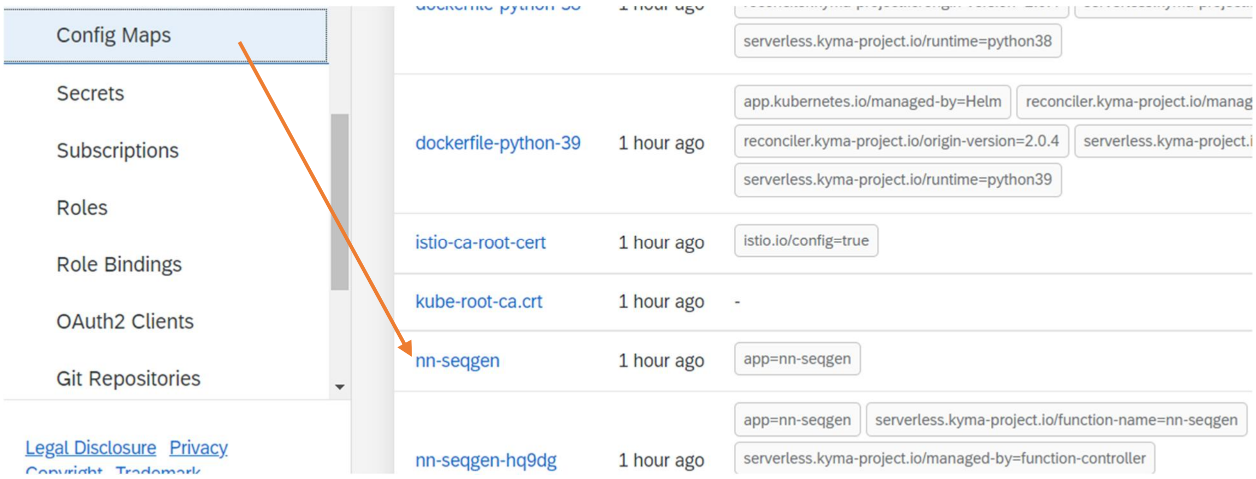
- Define Config Maps in nn-seqgen_configmap.yaml

Config Maps is used to store non-confidential data in key-value pairs.

Defines MongoDB database host and port. The host value assumes that the service for the MongoDB database is named as “mongo-nodeport-svc”.

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: nn-seqgen
 labels:
 app: nn-seqgen
data:
 host: mongo-nodeport-svc
 port: "27017"
```

The result of this deployment can be found if you navigate to Configuration à Config Maps in Kyma Console UI.



		serverless.kyma-project.io/runtime=python38
dockerfile-python-39	1 hour ago	app.kubernetes.io/managed-by=Helm reconciler.kyma-project.io/manag reconciler.kyma-project.io/origin-version=2.0.4 serverless.kyma-project.i serverless.kyma-project.io/runtime=python39
istio-ca-root-cert	1 hour ago	istio.io/config=true
kube-root-ca.crt	1 hour ago	-
nn-seqgen	1 hour ago	app=nn-seqgen
nn-seqgen-hq9dg	1 hour ago	app=nn-seqgen serverless.kyma-project.io/function-name=nn-seqgen serverless.kyma-project.io/managed-by=function-controller

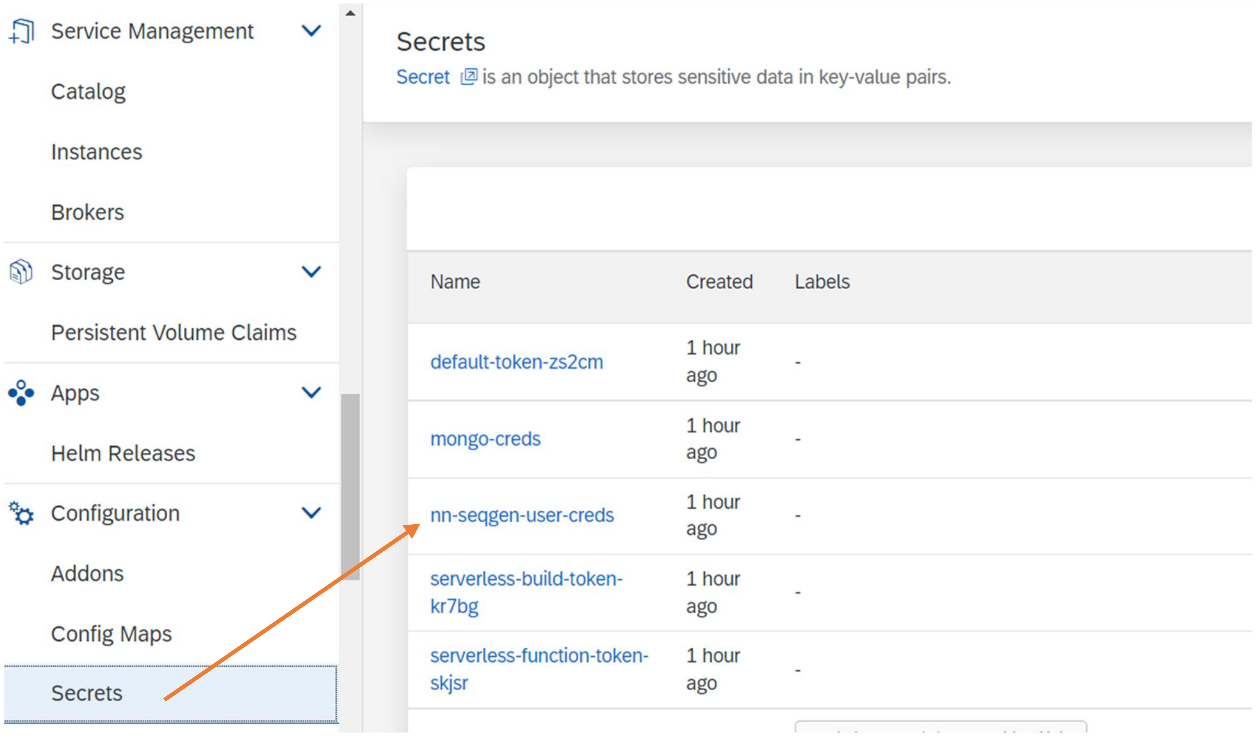
· Define MongoDB username and password map in nn-seqgen-secrets.yaml

Secret stores confidential data in key-value pairs, such as user name and password.

Defines the user name and password for "ident-db" MongoDB database. Secret values are base64-encoded.

```
apiVersion: v1
data:
  password: cGFzc3dvcmQxMjM0NTY3ODlA #password123456789@
  username: bm5fc2VxZ2VuX3VzZXI= #nn_seqgen_user
kind: Secret
metadata:
  name: nn-seqgen-user-creds
```

The result of this deployment can be found if you navigate to Configuration à Secrets in Kyma Console UI.



Secrets

Secret is an object that stores sensitive data in key-value pairs.

Name	Created	Labels
default-token-zs2cm	1 hour ago	-
mongo-creds	1 hour ago	-
nn-seqgen-user-creds	1 hour ago	-
serverless-build-token-kr7bg	1 hour ago	-
serverless-function-token-skjsr	1 hour ago	-

- Define nn-seqgen serverless function in nn-seqgen_function.yaml

“nn-seqgen” serverless function generates a new batch number based on environment variables: PATTERN, RESET_MODE, NUMBER_BASE. It uses Config Maps for getting MongoDB host and port settings. It uses Secrets for getting MongoDB user name and password.

```
apiVersion: serverless.kyma-project.io/v1alpha1
kind: Function
metadata:
 labels:
  app: nn-seqgen
 name: nn-seqgen
spec:
 deps: "{ \n   \"name\": \"nn-seqgen\",\n   \"version\": \"1.0.0\",\n   \"dependencies\":
  {\n      \"mongoose\": \"^6.0.0\"\n   }\n}"
 maxReplicas: 1
 minReplicas: 1
 resources:
  limits:
   cpu: 200m
   memory: 256Mi
  requests:
   cpu: 100m
   memory: 128Mi
 env:
  - name: PATTERN
   value: PLANTYYYYDDMMLLNNNNN
  - name: RESET_MODE
   value: DAY
  - name: NUMBER_BASE
```

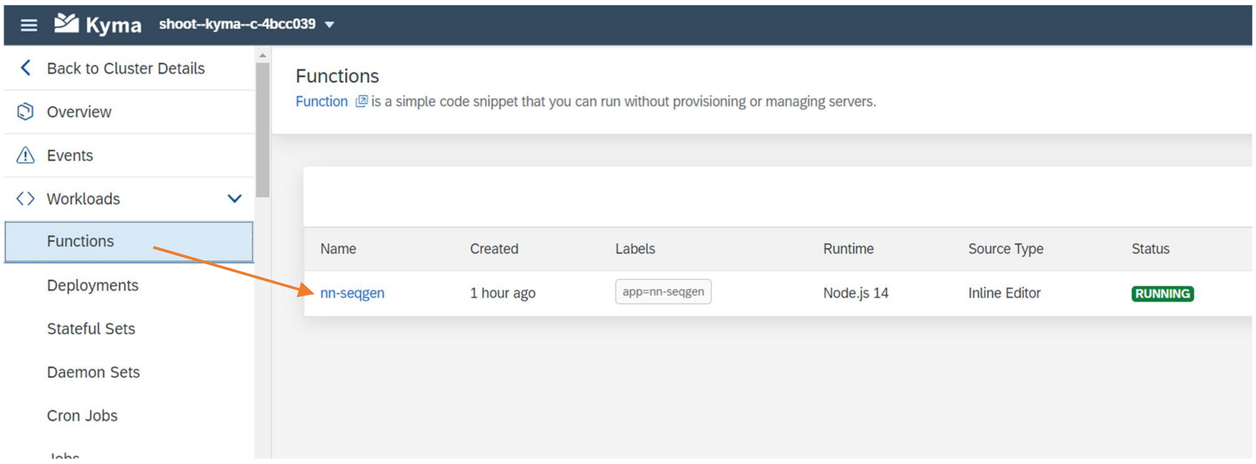
```
      value: "10"
    - name: MONGODB_HOST
      valueFrom:
        configMapKeyRef:
          key: host
          name: nn-seqgen
    - name: MONGODB_PORT
      valueFrom:
        configMapKeyRef:
          key: port
          name: nn-seqgen
    - name: MONGO_USER
      valueFrom:
        secretKeyRef:
          key: username
          name: nn-seqgen-user-creds
    - name: MONGO_PASSWORD
      valueFrom:
        secretKeyRef:
          key: password
          name: nn-seqgen-user-creds
  runtime: nodejs14
  source: "const mongoose = require('mongoose');\n// keep connection to MongoDB\nlet
    conn = null;\n// read host, port, user name, password from Environment Variables\nconst
    mongoDBHost = readEnv(\"MONGODB_HOST\");\nconst mongoDBPort =
readEnv(\"MONGODB_PORT\");\nconst
    userName = readEnv(\"MONGO_USER\");\n// password should be encoded \nconst password
    = encodeURIComponent(readEnv(\"MONGO_PASSWORD\"));\n// get reset mode for sequence,
    can be YEAR, MONTH, DAY, NONE \nconst resetMode = !isEmpty(readEnv(\"RESET_MODE\"))
    ? readEnv(\"RESET_MODE\") : \"NONE\";\n// get pattern for building identifier
    \nconst pattern = !isEmpty(readEnv(\"PATTERN\")) ? readEnv(\"PATTERN\") :
\"PLANTYYYYDDMMLLNNNNN\";\n//
    The number base of the sequence portion of the numbering pattern, supported 10
    for decimal and 16 for hexadecimal \nconst numberBase = !isEmpty(readEnv(\"NUMBER_BASE\"))
    ? readEnv(\"NUMBER_BASE\") : \"10\";\n// Define sequence schema \nconst sequenceSchema
    = mongoose.Schema({\n    sequence: {\n        type: Number,\n        default:
    0,\n    },\n    plant: String,\n    workCenter: String,\n    year: Number,\n    month:
    Number,\n    day: Number,\n    resetMode: String\n});\n\n//******************Extension
    Entry point ******************\nmodule.exports = {\n    main: async function(event,
    context) {\n        try {\n            // Because conn is in the global scope
    this means your Lambda function doesn't have to go through the\n            //
    potentially expensive process of connecting to MongoDB every time.\n            if
    (conn == null) {\n                //build connection string URL\n                let
    mongoDBConnectionURL =
`mongodb://${userName}:${password}@${mongoDBHost}:${mongoDBPort}/ident-
db?retryWrites=true`;\n
    \               //console.log('nn-seqgen mongoDBConnectionURL: ', mongoDBConnectionURL);\n
    \               conn = mongoose.createConnection(mongoDBConnectionURL, {\n
useNewUrlParser:
    true,\n                    useUnifiedTopology: true,\n                    serverSelectionTimeoutMS:
```

```
  5000\n            });\n            // await-ing connection after assigning
  to the conn variable\n            // to avoid multiple function calls creating
  new connections\n            await conn;\n        }\n        // if
  readyState = 1 then connection is okay\n        if (conn.readyState === 0)
  {\n            throw new Error(\"Could not connect to MongoDB.\");\n        }\n
  \        let request = event.data;\n        if (isEmpty(request)) {\n            throw
  new Error(\"No request body found.\");\n        }      \n        //console.log('nn-seqgen
  input request %j: ', request);\n        //we will collect the new identifiers
  here\n        let newIdentifiers = []; \n        let result;\n        if
  (\"identifiers\" in request) {\n            //console.log(\"nn-seqgen. Received
  Identifiers %j:\", request.identifiers);\n            result = await processIdentifiers(request.identifiers,
  newIdentifiers, \"extensionParameters\" in request ? request.extensionParameters
  : null);\n        } else {\n            throw new Error(\"No identifiers
  found in request\");\n        }\n        //console.log(\"nn-seqgen RESULT
  %j\", result);\n\n        request.identifiers = newIdentifiers;\n\n        //console.log(\"nn-seqgen
  returning request %j\", request);\n\n        return request;\n    } catch
  (err) {\n        console.error(\"an error occurred...\", err);\n
event.extensions.response.status(500).json({\n
  \            \"message\": \"An error occurred during nn-seqgen function execution\",\n
  \            \"error\": err.message\n        });\n    }\n  }\n}\n\n\n//******************
  processIdentifiers ****************************\n// Process all received identifiers
  and change them to defined format in
pattern\n//**********************************************************************\n\nasync
  function processIdentifiers(identifiers, newIdentifiers, extensionParameters)
  {\n  //console.log(\"nn-seqgen.ProcessIdentifiers %j , %j , %j \", identifiers,
  newIdentifiers, extensionParameters);\n  let workCenterParam;\n  let plant;\n
  \  let newIdentifier;\n  // read work center and plant value from extension
  parameters \n  if (extensionParameters) {\n    workCenterParam =
extensionParameters[\"WORK_CENTER\"];\n
  \    plant = extensionParameters[\"PLANT\"] ? extensionParameters[\"PLANT\"]
  : getPlantFromRouting(extensionParameters[\"ROUTING\"]);\n    //console.log(\"nn-seqgen
  workCenter Param\", workCenterParam);\n    //console.log(\"nn-seqgen plant
  Param\", plant);\n  } else {\n    throw new Error(\"No extension parameters
  found\");\n  }\n  if (!isEmpty(identifiers)) {\n    // create Model based
  on schema\n    const sequenceModel = conn.model('NextNumberSequence', sequenceSchema);\n
  \    for (let iter = 0; iter < identifiers.length; iter++) {\n        try
  {\n            newIdentifier = await buildNewIdentifier(sequenceModel, workCenterParam,
  plant);\n            // add new identifier\n            newIdentifiers.push(newIdentifier);\n
  \        } catch (err) {\n            //console.log(\"nn-seqgen.processIdentifiers:
  failed to process identifiers %j\", err);\n            throw err;\n        }\n
  \    }\n  }\n  return newIdentifiers;\n}\n\nasync function buildNewIdentifier(sequenceModel,
  workCenter, plant) {\n  let nextNum;\n  try {\n    nextNum = await
updateOrInsertSequence(sequenceModel,
  workCenter, plant);\n    //console.log('nn-seqgen.buildNewIdentifier', nextNum);\n
  \  } catch (err) {\n    //console.error(\"error calling buildNewIdentifier
  %j\", err);\n    throw err;\n  }\n  let newIdentifier = replacePatternPlaceholders(workCenter,
  plant, nextNum);\n  //console.log(\"nn-seqgen.new Identifier %s\", newIdentifier);\n
  \  return newIdentifier;\n}\n// replace placeholders in pattern string with actual
  values \nfunction replacePatternPlaceholders(workCenter, plant, nextNum) {\n  let
  newIndetifier = pattern;\n  let dateComposition = getDateComponents();\n  var
```

```
placeholders = {\n        \"YYYY\": dateComposition.fullYear,\n        \"YY\":
dateComposition.twoDigitYear,\n        \"MM\": dateComposition.month,\n        \"DD\":
dateComposition.day,\n        \"LL\": workCenter,\n        \"PLANT\": plant,\n
\        \"NNNNN\": numberToNumberBase(nextNum, numberBase, 5)\n    }\n    for
(var placeholder in placeholders) {\n        while (newIndetifier.indexOf(placeholder)
> -1) {\n            newIndetifier = newIndetifier.replace(placeholder, placeholders[placeholder])\n
\        }\n    }\n    return newIndetifier;\n\n}\n\n//update or Insert Sequence
document to collection\nasync function updateOrInsertSequence(sequenceModel, workCenter,
plant) {\n    let dateComposition = getDateComponents();\n    let filter;\n    //
incriment sequence on 1\n    let update = {\n        $inc: {\n            sequence:
1\n        }\n    };\n    // define search filter in collection \n    switch (resetMode)
{\n        case 'NONE':\n            filter = {\n                workCenter: workCenter,\n
\                plant: plant,\n                resetMode: resetMode\n            };\n
\            break;\n        case 'YEAR':\n            filter = {\n                workCenter:
workCenter,\n                plant: plant,\n                year: dateComposition.year,\n
\                resetMode: resetMode\n            };\n            break;\n        case
'MONTH':\n            filter = {\n                workCenter: workCenter,\n                plant:
plant,\n                month: dateComposition.month,\n                year: dateComposition.year,\n
\                resetMode: resetMode\n            };\n            break;\n        case
'DAY':\n            filter = {\n                workCenter: workCenter,\n                plant:
plant,\n                month: dateComposition.month,\n                year: dateComposition.year,\n
\                day: dateComposition.day,\n                resetMode: resetMode\n
\            };\n            break;\n        default:\n            filter = {\n
\                workCenter: workCenter,\n                plant: plant,\n                resetMode:
resetMode\n            };\n            break;\n    }\n    // upsert = true, means
that findOneAndUpdate method works like update or insert if not found\n    let
sequenceDocument = await sequenceModel.findOneAndUpdate(filter, update, {\n        new:
true,\n        upsert: true\n    });\n    //console.log(\"nn-seqgen document sequenceDocument:
%j\", sequenceDocument);\n    return sequenceDocument.sequence;\n}\n\n// read
Environment Variables\nfunction readEnv(sEnv) {\n    return process.env[sEnv];\n}\n\nfunction
getDateComponents() {\n    let today = new Date();\n    let dd = today.getDate().toString().padStart(2,
'0');\n    let mm = (today.getMonth() + 1).toString().padStart(2, '0');\n    let
yyyy = today.getFullYear();\n    let yy = today.getFullYear().toString().substr(-2);\n\n
\    return {\n        month: mm,\n        day: dd,\n        fullYear: yyyy,\n
\        twoDigitYear: yy\n    }\n}\n\nfunction isEmpty(obj) {\n    if (obj ==
null) return true;\n    // Assume if it has a length property with a non-zero
value that that property is correct.\n    if (obj.length && obj.length > 0) return
false;\n    if (obj.length === 0) return true;\n    for (var key in obj) {\n        if
(hasOwnProperty.call(obj, key)) return false;\n    }\n    return true;\n}\n//
extract Plant from Handle string\nfunction getPlantFromRouting(routing) {\n    //routing
input example:
\"RouterStepBO:RouterBO:KYMA,ROUTER1,U,A,10;RouterStepBO:RouterBO:KYMA,ROUTER1,U,A,20\"\n
\    // we need to extract plant, for example, KYMA \n    if (!isEmpty(routing))
{\n        var routingArr = routing.split(':');\n        if (!isEmpty(routingArr))
{\n            var routerBO = routingArr[2];\n            if (!isEmpty(routerBO))
{\n                var routerBOArr = routerBO.split(',');\n                if
(!isEmpty(routerBOArr)) {\n                    return routerBOArr[0];\n                }\n
\            }\n        }\n    }\n}\n// convert number to hex if needed and add
leading zeros. \nfunction numberToNumberBase(number, numberBase, padding) {\n
```

```
  \   if (numberBase === \"16\") {\n        //hexdecimal\n        number =
Number(number).toString(16).toUpperCase();\n
  \   }\n    //adding leading 0 to number\n    number = String(number).padStart(padding,
  '0');\n\n    return number;\n}"
```

The result of this deployment can be found if you navigate to Workloads à Functions in Kyma Console UI.

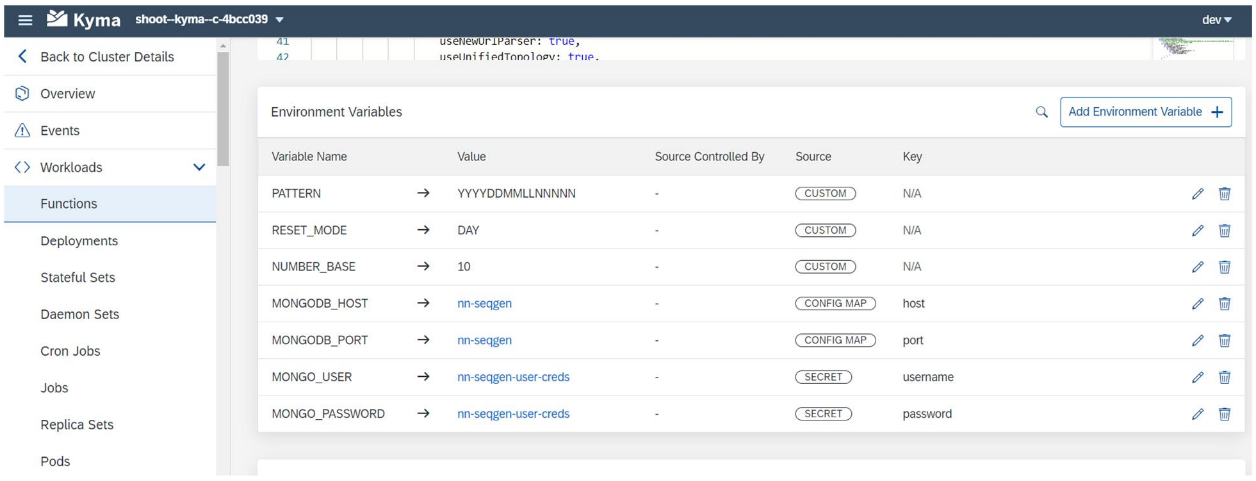


Functions

Function is a simple code snippet that you can run without provisioning or managing servers.

Name	Created	Labels	Runtime	Source Type	Status
nn-seqgen	1 hour ago	app=nn-seqgen	Node.js 14	Inline Editor	RUNNING

The function has Environment Variables



```
useNewUrlParser: true,
useUnifiedTopology: true,
```

Environment Variables

Variable Name		Value	Source Controlled By	Source	Key
PATTERN	→	YYYYDDMMLLNNNNN	-	CUSTOM	N/A
RESET_MODE	→	DAY	-	CUSTOM	N/A
NUMBER_BASE	→	10	-	CUSTOM	N/A
MONGODB_HOST	→	nn-seqgen	-	CONFIG MAP	host
MONGODB_PORT	→	nn-seqgen	-	CONFIG MAP	port
MONGO_USER	→	nn-seqgen-user-creds	-	SECRET	username
MONGO_PASSWORD	→	nn-seqgen-user-creds	-	SECRET	password

Here is how the YAML structure defines CUSTOM source environment variables. Such variables can be modified in a runtime.

```
 env:
   - name: PATTERN
     value: PLANTYYYYDDMMLLNNNNN
   - name: RESET_MODE
     value: DAY
   - name: NUMBER_BASE
     value: "10"
```

Here is how the YAML structure defines SECRET source environment variables. Such variables are read-only.

```
env:
  - name: MONGO_USER
    valueFrom:
      secretKeyRef:
        key: username
        name: nn-seqgen-user-creds
  - name: MONGO_PASSWORD
    valueFrom:
      secretKeyRef:
        key: password
        name: nn-seqgen-user-creds
```

Here is how the YAML structure defines CONFIG MAP source environment variables. Such variables are read-only.

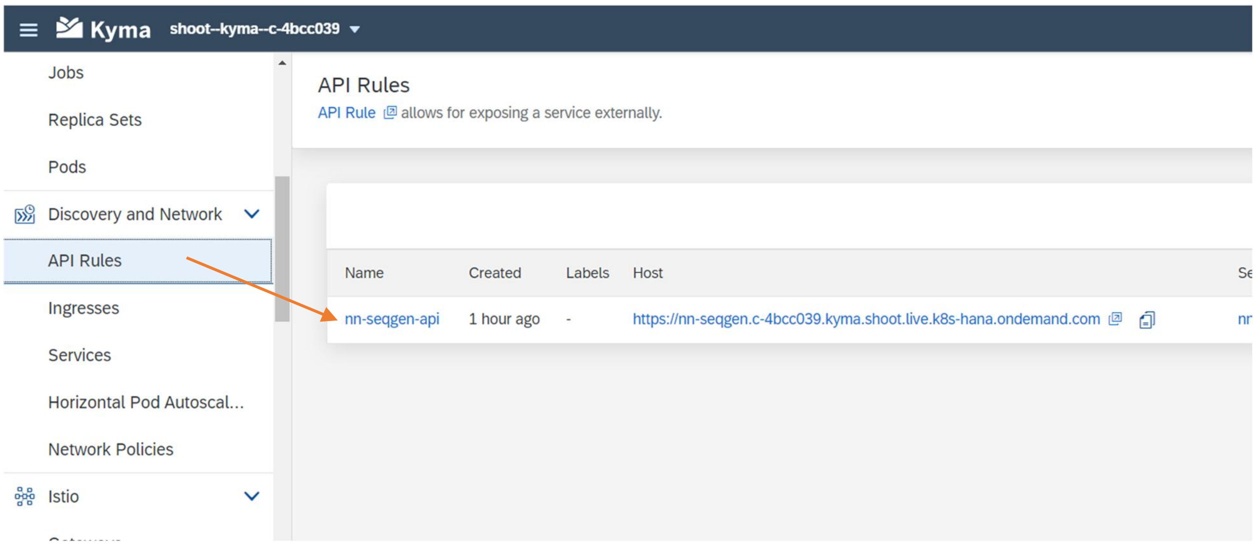
```
env:
  - name: MONGODB_HOST
    valueFrom:
      configMapKeyRef:
        key: host
        name: nn-seqgen
  - name: MONGODB_PORT
    valueFrom:
      configMapKeyRef:
        key: port
        name: nn-seqgen
```

- Define API Rule in nn-seqgen_apirule.yaml

Defines the API endpoint, which exposes the services outside the cluster.

```
apiVersion: gateway.kyma-project.io/v1alpha1
kind: APIRule
metadata:
 name: nn-seqgen-api
spec:
 gateway: kyma-gateway.kyma-system.svc.cluster.local
 rules:
  - accessStrategies:
      - config: {}
        handler: noop
    methods:
      - POST
    path: /.*
 service:
  host: nn-seqgen
  name: nn-seqgen
  port: 80
```

The result of this deployment can be found if you navigate to Discovery and Network à API Rules in Kyma Console UI.
Use Host value as URL path for defining new service in the Manage Service Registry DME application.
This new service can be used as an in-App Extension for Next Number functionality.



API Rules

API Rule allows for exposing a service externally.

Name	Created	Labels	Host	Se
nn-seqgen-api	1 hour ago	-	https://nn-seqgen.c-4bcc039.kyma.shoot.live.k8s-hana.ondemand.com	nr