in this document:

- **Why API?**
- **Simple YouTube-style embedding of simulation animations**
- **General information**
- **JavaScript API**
- **Java API**
- **Features available in AnyLogic Private Cloud only**
why API?

- **AnyLogic Cloud web UI** enables you to do a lot of things
  - Perform highly parallel multi-run experiments
  - Run interactive animated simulations in the browser
  - Setup experiments inputs and configure dashboards
  - Download results
  - Manage versions
  - Share models and collaborate

- **AnyLogic Cloud API** further extends this functionality. It allows you to:
  - Build custom web interfaces to cloud-based models with optional embedding of model animations into your web page (**JavaScript API**)
  - Use cloud-based models as a part of a custom analytical workflow by programmatically setting up and invoking single- and multiple-run simulations, and retrieving outputs (**Java API**)

![AnyLogic Cloud web UI](image)

![AnyLogic Cloud API](image)
Before we dive into the API:
Simple YouTube-style embedding of simulation animations
(available for anyone!)
If all you need is just have the AnyLogic Cloud model animation on your HTML page — you can embed it just like a YouTube video:

1. Display the model tile (either in the tiles view or in the model page)
2. Click the “<>” icon at the bottom right corner of the tile
3. Copy the HTML fragment displayed in the dialog box
4. Paste the fragment into your HTML page where you want the animation to appear:

```html
<head>
  ...
</head>
<body>
  ...
  <iframe width="1000" height="650" src="https://cloud.anylogic.com/assets/embed/modelId=fac9553-d749-4008-9272-0e46c2e5122a" allowfullscreen/>
  ...
</body>
```
restrictions of simple embedding

• Only animations of public models can be embedded that way

• The animation always refers to the same simulation as the public page of the model, i.e.
  – ... to a simulation experiment of the latest model version will run
  – If the latest version has no simulation experiments, an error will occur

• Keep in mind:
  – You will only be able to control the model execution from the default animation toolbar
  – You will not be able to change parameters prior to simulation or save outputs afterwards
  – When the Stop button is pressed, the animation returns to the start page
General information on AnyLogic Cloud API
AnyLogic Cloud API works over HTTP connection

- Regardless the language used, the API translates function calls into HTTP requests that go to the AnyLogic Cloud server, waits and returns the responses
  - A typical HTTP request/response sequence:
AnyLogic Cloud endpoints

- It is recommended to use the API clients provided by AnyLogic. In case you plan to implement your own client here are the endpoints*

<table>
<thead>
<tr>
<th>Root address**</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://cloud.anylogic.com/api/open/8.3">https://cloud.anylogic.com/api/open/8.3</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>URL</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get all user models</td>
<td>GET /models</td>
<td>Model[]</td>
</tr>
<tr>
<td>Get model by id</td>
<td>GET /models/&lt;id&gt;</td>
<td>Model</td>
</tr>
<tr>
<td>Get model by name</td>
<td>GET /models/name/&lt;modelName&gt;</td>
<td>Model</td>
</tr>
<tr>
<td>Get model version by number</td>
<td>GET /models/&lt;id&gt;/versions/number/&lt;versionNumber&gt;</td>
<td>ModelVersion</td>
</tr>
<tr>
<td>Get model version by id</td>
<td>GET /models/&lt;id&gt;/versions/&lt;versionId&gt;</td>
<td>ModelVersion</td>
</tr>
<tr>
<td>Get model version experiments</td>
<td>GET /versions/&lt;versionId&gt;/experiments</td>
<td>Experiment[]</td>
</tr>
<tr>
<td>Start model run</td>
<td>POST /versions/&lt;versionId&gt;/runs body: RunRequest</td>
<td>RunState</td>
</tr>
<tr>
<td>Stop model run</td>
<td>POST /versions/&lt;versionId&gt;/runs/stop body: RunRequest</td>
<td>RunState</td>
</tr>
<tr>
<td>Start animated model run</td>
<td>POST /versions/&lt;versionId&gt;/runs/animation body: RunRequest</td>
<td></td>
</tr>
<tr>
<td>Get model run</td>
<td>POST /versions/&lt;versionId&gt;/run body: RunRequest</td>
<td>RunState</td>
</tr>
<tr>
<td>Get run outputs</td>
<td>POST /versions/&lt;versionId&gt;/runs/&lt;runId&gt; body: blank Outputs</td>
<td>Outputs</td>
</tr>
</tbody>
</table>

* You must provide “Authorization: <apiKey>” header in every HTTP request (see API Key)

** This is the root address of the AnyLogic Cloud for subscribers. In case of AnyLogic Private Cloud the address will be different.
authentication with API Key

• API Key is a randomly generated personal identifier you need to access AnyLogic Cloud API
  – This is an example API Key:
    
    e05a6efa-ea5f-4adf-b090-ae0ca7d16c20

• To obtain the API Key (subscribers and Private Cloud users* only):
  – Login to AnyLogic Cloud with web UI
  – Go to your Profile page
  – Expand the API Key section
  – The key is here:

    ![API Key Interface]

    You can generate a new key if the existing one has been exposed

  – Keep the key safe! If it gets exposed, generate a new key

* In a Private Cloud, every user has his own API key
AnyLogic Cloud objects you will work with

My Models:
- model A
- model B
- model C
- model D

Versions of Model C:
1. version 1
2. version 2
3. version 3
4. version 6

Input/output pairs stored for performed simulations of Model C version 6 (directly accessible by Private Cloud users only):

Experiments defined in web UI:
- exper S1
- exper MC
- exper S2
- exper PV

Simulations to be or being performed right now (e.g. invoked by the API)
Collectively:

Simulations for these input sets (referred by experiments) have not yet been performed
JavaScript/Java object examples in JSON format (1/2)

```json
{  
    "id": "f646f6ec-4dc8-40f2-b8e3-6ba3c0de5s1a",
    "name": "Consumer Credit",
    "description": "This is a simulation of process...",
    "published": true,
    "modelVersions": [
        "19e85195-fdcd-4932-95fa-7a7d7866de43",
        "8f5b944a-be1b-4e1f-9da4-e70bc1fb48fc",
        "3a6dd161-a4f5-409d-bc2b-66db0658825",
        "a843326b-3df1-49a7-bc2f-a950fc978ef8"
    ]
}
```

```json
{  
    "id": "a843326b-3df1-49a7-bc2f-a950fc978ef8",
    "version": "4",
    "experimentTemplate": {  
        "inputs": [  
            {  
                "name": "{STOP_MODE}",
                "type": "STRING",
                "units": null,
                "value": "STOP_MODE_AT_TIME"
            },
            {  
                "name": "Rate",
                "type": "DOUBLE",
                "units": null,
                "value": "11.5"
            }
        ],
        "outputs": [  
            ...
        ],
        "dashboard": [  
            ...
        ]
    }
}
```

A system parameter

© The AnyLogic Company | www.anylogic.com
JavaScript/Java object examples in JSON format (2/2)

```javascript
// Experiment
{
  "id": "d2496ecd-7f1e-4556-a70e-c6c4e826e8a7",
  "name": "Experiment",
  "type": "SIMULATION",
  "modelVersion": "a843326b-3df1-47a7-bc2f-a950fa978ef8",
  "inputs": [
    {
      "name": "Rate",
      "type": "DOUBLE",
      "units": null,
      "value": "11.5"
    },
    {
      "name": "Capacity",
      "type": "INTEGER",
      "units": null,
      "value": "23"
    },
    {
      "name": "{MAX_MEMORY_MB}"
      "type": "INTEGER",
      "units": null,
      "value": "512"
    }
  ]
}

// Outputs
List<ModelData> Single run
[
  {
    "name": "Mean time in system",
    "type": "DOUBLE",
    "units": "HOUR",
    "value": "7.0"
  },
  {
    "name": "Application life cycle",
    "type": "HISTOGRAM_DATA",
    "units": null,
    "value": "{statistics:{"type":"DISCRETE, ...}}"
  }
]

// Outputs
List<ModelData> Multiple runs
[
  {
    "name": "Mean time in system",
    "type": "DOUBLE",
    "units": "HOUR",
    "value": "[7.0,8.0,9.0,10.0]"
  },
  {
    "name": "Application life cycle",
    "type": "HISTOGRAM_DATA",
    "units": null,
    "value": "[\{statistics: ...\}, ...]"
  }
]
```

In multiple runs outputs, every value becomes an array of values.
JavaScript API
for building custom web interfaces to cloud-based models
(with and without embedding model animation)
AnyLogic Cloud JavaScript client

• Use JavaScript client to build custom web interfaces to cloud-based AnyLogic models

• Do the following:
  2. Unpack the .zip file. You will see this directory structure:
  3. Copy the content of the “assets” folder into your website’s “assets” directory
  4. Load the cloud_client library by adding the <script> tag to the head of your webpage*: 
     <script src="assets/cloud_client.js"></script>
  5. Obtain the API key in the AnyLogic Cloud web UI.
  6. Create a new instance of the CloudClient object
     cloudClient = CloudClient.create( <apiKey> ); or 
     cloudClient = CloudClient.create( <apiKey>, <URL of your private cloud> );
  7. Use methods of CloudClient and ModelRun objects to work with cloud-based models

• See the following slides for the JavaScript API usage examples
  – The JavaScript API is asynchronous and extensively uses the Promise technology
  – Check out also index.html and main.js located in the “js-client-8.3.0” directory

* Make sure your IDE compiler settings allow ECMAScript 6 JavaScript version
code example: simulation run with animation

```html
<html lang="en">
<head>
  <script src="assets/cloud_client.js"></script>
</head>
<body>
  <button onclick="runAnimation()">Run animation</button>
  <div id="animation-container" style="width: 800px; height: 600px">
    <!- the Client will insert animation player here like this:>
    <div id="..." style="width: 100%; height: 100%;">...<div>
  </div>
</body>
</html>
```

```javascript
let cloudClient = CloudClient.create( "e05a6efa-ea5f-4adf-b090-ae0ca7d16c20" );

function runAnimation() {
  cloudClient.getModelByName( "Bass Diffusion Demo" )
    .then( model => cloudClient.getLatestModelVersion( model ) )
    .then( version => {
      modelRun = cloudClient.createAnimation( version, "animation-container" );
      modelRun.setInput( "Contact Rate", 30 );
      return cloudClient.run( modelRun );
    })
    .catch( error => alert( "Unable to run animation. Error: " + error ) );
}
```
**code example: simulation run without animation**

**HTML**
```html
<head>
  <!-- Add this script into page header -->
  <script src="assets/cloud_client.js" defer></script>
</head>
<body>
  ...
  <!-- Simulation is set up and controlled from your custom UI -->
  <!-- No animation is shown -->
  ...
</body>
```

**JavaScript**
```javascript
let cloudClient = CloudClient.create( "e05a6efa-ea5f-4adf-b090-ae0ca7d16c20" );

function runAnimation() {
  cloudClient.getModelByName( "Bass Diffusion Demo" )
    .then( model => cloudClient.getLatestModelVersion(model) )
    .then( version => {
      modelRun = cloudClient.createSimulation(version);
      modelRun.setInput( "Contact Rate", 40 );
      return cloudClient.run( modelRun );
    })
    .then( outputs => { 
      let ap = outputs.find( output => output.name === "AdoptionPercent" ).value;
      console.log( "AdoptionPercent = " + ap );
    })
    .catch( error => alert( "Unable to run simulation. Error: " + error ) );
  }
```
code example: parameter variation

```javascript
let cloudClient = CloudClient.create( "e05a6efa-ea5f-4adf-b090-ae0ca7d16c20" );

function runParameterVariation() {
    cloudClient.getModelByName( "Bass Diffusion Demo" )
        .then( model => cloudClient.getModelVersionByNumber( model, 2 ) )
        .then( version => {
            modelRun = cloudClient.createParameterVariation(version);
            modelRun.setInput( "Adoption Fraction", 0.02 );
            modelRun.setRangeInput( "Contact Rate", 20, 180, 20 );
            return cloudClient.run( modelRun );
        })
        .then( outputs => {
            let ap = outputs.find( output => output.name == "AdoptionPercent" ).value;
            console.log( "AdoptionPercent: " + ap );
        })
        .catch( error => alert( "Unable to run parameter variation. Error: " + error ) );
}
```

```html
<head>
    <!-- Add this script into page header -->
    <script src="assets/cloud_client.js" defer></script>
</head>
<body>
    ...
    <!-- Simulation is set up and controlled from your custom UI -->
    <!-- No animation is shown -->
    ...
</body>
```
code example: three parallel simulations

**HTML** is the same as in the **parameter variation example**

**JavaScript**

```javascript
let cloudClient = CloudClient.create( "e05a6efa-ea5f-4adf-b090-ae0ca7d16c20" );

function runThreeParallelSimulations() {
    let crValues = [ 20, 50, 100 ]; //three values for Contact Rate parameter
    cloudClient.getModelByName( "Bass Diffusion Demo" )
        .then( model => cloudClient.getLatestModelVersion(model) )
        .then( version => {
            let tasks = []; //this will be array of Promises
            //launch 3 runs in parallel
            for( cr of crValues ) {
                modelRun = cloudClient.createSimulation( version );
                modelRun.setInput( "Contact Rate", cr );
                tasks.push( cloudClient.run( modelRun ) ); //add run Promise to the array
            }
            return Promise.all( tasks ); //this waits for ALL promises to complete
        })
        .then( outputsArray => {
            //we now have array of three Output objects (order is kept)
            for( let i = 0; i < crValues.length; i++ ) {
                //let's find and print some particular output and the corresponding input
                let ap = outputsArray[i].find( output => output.name == "AdoptionPercent" ).value;
                console.log( "Contact Rate = " + crValues[i] + " --> " +
                        "Adoption Percent = " + ap );
            }
        })
        .catch( error => alert( "Unable to run three parallel simulations. Error: " + error ) );
```

© The AnyLogic Company | www.anylogic.com
class CloudClient

static create(apiKey) returns CloudClient — A static function that creates a CloudClient object, given the API Key.

getModels() returns Model[] — Returns the array of models listed in the My Models section of the AnyLogic Cloud web UI.

getModelById(id) returns Promise* Model — Finds and returns (by promise) a model with a given ID.

getModelByName(name) returns Promise Model — Finds and returns (by promise) a model with a given name.

getModelVersionById(model, versionId) returns Promise ModelVersion — Finds and returns (by promise) a model version with a given ID.

getModelVersionByNumber(model, number) returns Promise ModelVersion — Finds and returns (by promise) a model version with a given number.

getLatestModelVersion(model) returns Promise ModelVersion — Finds and returns (by promise) the latest model version.

getModelVersionExperiments(modelVersion) returns Promise Experiment[] — Returns (by promise) the array of experiments defined in the web UI for a given model version. Experiments may be used to set inputs of model runs.

createSimulation(modelVersion) returns ModelRun — creates and returns a ModelRun objects describing a simulation to be performed.

createParameterVariation(modelVersion) returns ModelRun — creates and returns a ModelRun objects describing a parameter variation to be performed.

createAnimation(modelVersion, divId) returns ModelRun — creates and returns a ModelRun objects describing a simulation to be performed with animation displayed in a <div> with a given ID.

run(ModelRun) returns Promise Outputs — Starts a given run. Returns (by promise) the run outputs, or empty outputs in case of animation run.

stop(ModelRun) returns Promise ModelRun — Requests to stop a given run and returns (by promise) the stopped run.

class ModelRun

setInput(name, value) – Sets the value of a given input in the run.*

setRangeInput(name, min, max, step) – Sets the variation range for a given input.

setInputsFromExperiment(experiment) – Copies all input values from a given experiment to the run. Experiment can be obtained from the cloud client by calling its getModelVersionExperiments() method.

* If you do not set an input value, the default value defined in the model is used
JavaScript API upcoming features:

- MonteCarlo runs construction
- Requesting model run state / progress
- Control of animated model from API: stop/run/pause/change parameter from custom HTML (outside the animation box)
  - Hiding the default toolbar sections (Run, Speed, etc.) – this is done in AnyLogic model programmatically
  - Pause/resume
  - Navigation
  - Changing a parameter at runtime
  - (Please note that all controls in animation will still work)
Java API
for using cloud-based AnyLogic models in custom analytical workflows
• Java API is primarily a backend technology that allows you to invoke cloud-based AnyLogic simulations from Java code and thus integrate simulations into custom workflows

• Java API assumes fast, animation-less model execution
  – Should you need to display animation, use HTML and JavaScript API

• Java API works over HTTP API in the same way the JavaScript API does and it is similar to JavaScript API in terms of usage scenarios, objects and methods

• There are two types of Java API:
  – Synchronous API where methods wait for completion of underlying HTTP requests (and block the thread where they are called)
  – Asynchronous API where methods do not wait for completion of HTTP requests (and thus do not block the thread) and return CompletableFuture objects instead
  – Synchronous API is easier to use than asynchronous, but asynchronous API is better suited for certain use cases (like launching a customized set of simulations in parallel)
AnyLogic Cloud Java client

• Use Java client to integrate cloud-based AnyLogic models into custom analytical workflows

• Do the following:
  2. Unpack the .zip file. You will see this directory structure:
  3. Copy the file “anylogic-cloud-client-8.3.0.jar” located in the “java-client-8.3.0” folder to your Java project, e.g. to the “lib” folder (you can also find Javadoc and source files in the same location)
  4. Add the file “anylogic-cloud-client-8.3.0.jar” as a library to your project (and optionally, you can copy and attach the corresponding Javadoc file)
  5. Obtain the API key in the AnyLogic Cloud web UI.
  6. Create an instance of the appropriate Java client (AnyLogicCloudClient or AsyncAnyLogicCloudClient) and use its methods to access cloud-based models and run simulations

• See the following slides for synchronous and asynchronous API usage examples
  – You can also find these examples in the Example.java file located in the same folder
code example (synchronous): a single simulation run

Java

```java
import com.anylogic.cloud.clients.client_8_3_0.*;
import com.anylogic.cloud.service.open_8_3_0.api.project.*;

import java.util.List;

public class Main {

    public static void main( String[] args ) {
        AnyLogicCloudClient client =
            new AnyLogicCloudClient( "e05a6efa-ea5f-4adf-b090-ae0ca7d16c20" );
        Model model = client.getModelByName( "Bass Diffusion Demo" );
        ModelVersion modelVersion = client.getLatestModelVersion( model );
        ModelRun simulation = client.createSimulation( modelVersion );
        //set one of the inputs; other inputs will remain at default values
        simulation.setInput( "Contact Rate", 40 );
        List<ModelData> outputs = client.run( simulation ); //will exit upon run completion!
        System.out.println( outputs ); //this prints all outputs*
    }
}
```

* See JavaScript/Java object examples for the structure of run outputs (List<ModelData>)
code example (synchronous): parameter variation

```java
import com.anylogic.cloud.clients.client_8_3_0.*;
import com.anylogic.cloud.service.open_8_3_0.api.project.*;
import com.anylogic.cloud.service.open_8_3_0.api.project.data.ModelData;

import java.util.List;

public class Main {

    public static void main( String[] args ) {

        AnyLogicCloudClient client =
            new AnyLogicCloudClient( "e05a6efa-ea5f-4adf-b090-ae0ca7d16c20" );
        Model model = client.getModelByName( "Bass Diffusion Demo" );
        ModelVersion modelVersion = client.getLatestModelVersion( model );
        ModelRun variation = client.createParameterVariation( modelVersion );
        //set an input to range 20..180 with step 20; this will result in 9 single runs
        variation.setRangeInput( "Contact Rate", 20, 180, 20 );
        List<ModelData> outputs = client.run( variation ); //will exit upon run completion!
        ModelData ap =
            outputs.stream().
                filter( md -> md.name.equals( "AdoptionPercent" ) ).
                findFirst().orElse( null ); //this finds a particular output*
        System.out.println( ap );
    }
}

* See JavaScript/Java object examples for the structure of run outputs (List<ModelData>)
```
import com.anylogic.cloud.clients.client_8_3_0.*;
import com.anylogic.cloud.service.open_8_3_0.api.project.*;
import com.anylogic.cloud.service.open_8_3_0.api.project.data.ModelData;
import java.util.List;

public class Main {

    public static void main( String[] args ) {

        AsyncAnyLogicCloudClient asyncClient =
            new AsyncAnyLogicCloudClient( "e05a6efa-ea5f-4adf-b090-ae0ca7d16c20" );

        asyncClient.getModelByName( "Bass Diffusion Demo" )
            .thenCompose( asyncClient::getLatestModelVersion )
            .thenAccept( mv -> {

                int[] crValues = { 20, 50, 100 }; //array of values for a parameter
                CompletableFuture< List<ModelData> >[] tasks =
                    new CompletableFuture[ crValues.length ]; //array of future outputs
                //launch 3 runs in parallel
                for( int i = 0; i < crValues.length; i++ ) {
                    ModelRun simulation = asyncClient.createSimulation(mv);
                    simulation.setInput( "Contact Rate", crValues[i] );
                    //start a single run asynchronously, remember future outputs
                    tasks[i] = asyncClient.run(simulation); //exits immediately!
                }
            }

        continued on the next page >>
    }
}
code example (asynchronous): three parallel runs (2/2)

```java
>> continued from previous page

try {
    for (int i = 0; i < crValues.length; i++) {
        // wait for the run to complete: get() exits upon completion!
        List<ModelData> outputs = tasks[i].get();
        // find some particular output
        String apValue = outputs.stream()
            .filter(md -> md.name.equals("AdoptionPercent"))
            .findFirst()
            .map(md -> md.value)
            .orElse(null);

        System.out.println("Contact Rate = " + crValues[i] + " --> " +
                          "Adoption Percent = " + apValue);
    }
} catch (Exception e) {
    e.printStackTrace();
}
}
```

> whenComplete( (r, e) -> asyncClient.terminate() ); // terminate in any case
class AnyLogicCloudClient (synchronous API)

- This class implements CloudClient interface
- This is a synchronous client. Its methods wait for the completion of HTTP requests (and thus block the thread) before returning the result

AnyLogicCloudClient(String apiKey) — Creates a synchronous AnyLogic cloud client with a given API key and the default public cloud host name "https://cloud.anylogic.com".

AnyLogicCloudClient(String apiKey, String host) — Creates a synchronous AnyLogic cloud client with a given API key and a given host name (the latter makes sense in private clouds).
interface **CloudClient** (synchronous API)

- `List<Model> getModels()` — Returns the list of models listed in the My Models section of the AnyLogic Cloud web UI.
- `Model getModelById(String id)` — Finds and returns the model with a given ID.
- `Model getModelByName(String name)` — Finds and returns the model with a given name.
- `ModelVersion getModelVersionById(Model model, String versionId)` — Finds and returns the model version with a given ID.
- `ModelVersion getModelVersionByNumber(Model model, String versionNumber)` — Finds and returns the model version with a given number.
- `ModelVersion getLatestModelVersion(Model model)` — Returns the latest version of a given model.
- `List<Experiment> getModelVersionExperiments(ModelVersion modelVersion)` — Returns the list of experiments defined in the web UI for a given model version. Experiments may be used to set inputs of model runs.
- `ModelRun createSimulation(ModelVersion modelVersion)` — Creates a `ModelRun` objects describing a simulation to be performed.
- `ModelRun createParameterVariation(ModelVersion modelVersion)` — Creates a `ModelRun` objects describing a parameter variation to be performed.
- `List<ModelData> run(ModelRun modelRun)` — Starts the model run and waits for the run completion. Polling is used to determine the run state. The thread is blocked until the run is completed or terminated. The `modelRun` should be prepared to be executed (inputs should be set up).
- `ModelRun stop(ModelRun modelRun)` — Requests to stop a given run and waits for stop.
class AsyncAnyLogicCloudClient (asynchronous API)

- This class implements AsyncCloudClient interface
- This is an asynchronous client. Its methods do not wait for the completion of HTTP requests (and thus do not block the thread) and return CompletableFuture* objects instead

AnyLogicCloudClient(String apiKey) — Creates an asynchronous AnyLogic cloud client with a given API key and the default public cloud host name "https://cloud.anylogic.com".

AnyLogicCloudClient(String apiKey, String host) — Creates an asynchronous AnyLogic cloud client with a given API key and a given host name (the latter makes sense in private clouds).

* See e.g. https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html to learn how to work with CompletableFuture
interface AsyncCloudClient (asynchronous API)

CompletableFuture<List<Model>> getModels() — Asynchronously returns the list of models listed in the My Models section of the AnyLogic Cloud web UI.

CompletableFuture<Model> getModelById(String id) — Asynchronously finds and returns the model with a given ID.

CompletableFuture<Model> getModelByName(String name) — Asynchronously finds and returns the model with a given name.

CompletableFuture<ModelVersion> getModelVersionById(Model model, String versionId) — Asynchronously finds and returns the model version with a given ID.

CompletableFuture<ModelVersion> getModelVersionByNumber(Model model, String versionNumber) — Asynchronously finds and returns the model version with a given number.

CompletableFuture<ModelVersion> getLatestModelVersion(Model model) — Asynchronously returns the latest version of a given model.

CompletableFuture<List<Experiment>> getModelVersionExperiments(ModelVersion modelVersion) — Asynchronously returns the list of experiments defined in the web UI for a given model version. Experiments may be used to set inputs of model runs.

ModelRun createSimulation(ModelVersion modelVersion) — Creates a ModelRun objects describing a simulation to be performed.

ModelRun createParameterVariation(ModelVersion modelVersion) — Creates a ModelRun objects describing a parameter variation to be performed.

CompletableFuture<List<ModelData>> run(ModelRun modelRun) — Starts the model run. The modelRun should be prepared to be executed (inputs should be set up). Asynchronously waits for the run completion and returns the list of outputs.

CompletableFuture<ModelRun> stop(ModelRun modelRun) — Requests to stop a given run. Asynchronously waits for the run to stop and returns the run.

void terminate() — Terminates the client and kills all threads associated with it, so that no callbacks will be executed. Must be called when the client is no longer needed. The model runs initiated by the client, if any, will continue running in the cloud until completion.
class ModelRun

- Use createSimulation and createParameterVariation methods to create ModelRun objects

List<ModelData> getInputs() – Returns the list of model inputs backed by the model run (not a copy). Any changes made in the list will have the same effect as calling one of setInput methods.

void setInput(String name, int value) – Sets the value of a given integer input.*
void setInput(String name, long value) – Sets the value of a given long integer input.*
void setInput(String name, double value) – Sets the value of a given (double precision) real input.*
void setInput(String name, String value) – Sets the value of a given string input.*
void setRangeInput(String name, double min, double max, double step) – Sets the variation range for a given input.
void setInputsFromExperiment(Experiment experiment) – Copies all input values from a given experiment to the run. Experiment can be obtained from cloud client by calling its getModelVersionExperiments() method.

* If you do not set an input value, the default value defined in the model is used
Java API upcoming features:

- MonteCarlo runs construction
- Requesting model run state / progress
features available in AnyLogic Private Cloud only
features available in AnyLogic Private Cloud only

• The Private Cloud users are able to directly access the internal database where the users, models, and data are stored. Therefore, in the Private Cloud you can e.g.:
  – Directly access and manage users, models, versions, etc.
  – Access and search in the input/output pairs stored for performed simulations
  – Perform custom aggregation of outputs for a range of simulations

• In the Private Cloud, you will be able to scale the simulation on as many nodes as you want
  – (Subscribers to the cloud provided by AnyLogic run their simulations in a predefined pool of nodes)

• Finally, we will support your development of custom workflow cases
  – And help you, should you need access to the API not described in this document
Have a question on AnyLogic Cloud API?

- Use **Get Support** button in AnyLogic desktop application

- Or **AnyLogic Support** form in AnyLogic Cloud
Thank you!

- Please help us to make AnyLogic Cloud API better – let us know what else would you like to have there!