LifeTime Deployment API Examples

We provide real-world scenarios of using several API calls to perform common tasks like:

- Creating an application version;
- Deploying an application.

Create a new Application Version

In this example we will perform the following generic steps using API calls:

1. Get the environment key of the environment where you want to create the application version *(optional if you already have this information)*;

2. Get the application key of the desired application *(optional if you already have this information)*;

3. Get the module version key of each application module;

4. Create a new application version using the module(s) version information obtained in the previous step.

Check below for example API calls (requests and responses) for each of the presented steps.

1. Get environment key

We start by calling the API method that returns all the environments available on your infrastructure (only necessary if we don't know the environment key yet):

Request: GET /lifetimeapi/rest/v2/environments/

Response body:
Since we want to create an application version in our environment of type "Development" — coincidentally, the environment is also called "Development" —, we take note of its key so that we can use it in the upcoming API calls:

Environment key: f3582e43-43c7-4bb4-8cbb-d9f6c6bd35e

2. Get application key

Next, we call the API method that returns all the available applications in the infrastructure to get the application key (only necessary if we don't know the application key yet):

Request: **GET /lifetimeapi/rest/v2/applications/**

Response body:

```json
[  
   {  
      "Key": "f3582e43-43c7-4bb4-8cbb-d9f6c6bd35e",
      "Name": "Development",
      "OSVersion": "11.0.110.0",
      "Order": 0,
      "HostName": "dev-env.company.com",
      "UseHTTPS": true,
      "EnvironmentType": "Development",
      "NumberOfFrontEnds": 1,
      "ApplicationServerType": ".NET",
      "ApplicationServer": "IIS",
      "DatabaseProvider": "SQLServer",
      "IsCloudEnvironment": false  
   },
   {  
      "Key": "849515f2-b4ff-4aca-a9d6-9407bea655f4",
      "Name": "Production",
      "OSVersion": "11.0.110.0",
      "Order": 1,
      "HostName": "prd-env-1.company.com",
      "UseHTTPS": true,
      "EnvironmentType": "Production",
      "NumberOfFrontEnds": 1,
      "ApplicationServerType": ".NET",
      "ApplicationServer": "IIS",
      "DatabaseProvider": "SQLServer",
      "IsCloudEnvironment": false  
   }  
]  
```
We want to create an application version for the "EmployeeBackoffice" application. We take note of its key so that we can use it in the next API calls:

"EmployeeBackoffice" application key: c9a7a82e-0eee-4a3d-8e22-2a19c69c766f

3. Get module version key of each application module

We start by taking the API method "template" URL used to get the details of an application on a given environment:

GET /lifetimeapi/rest/v2/environments/{EnvironmentKey}/applications/{ApplicationKey}/

We then change the {EnvironmentKey} and {ApplicationKey} placeholders to the correct values and perform the following request:

Request: GET /lifetimeapi/rest/v2/environments/f3582e43-43c7-4bb4-8cbb-d9f6cbbc35e/applications/c9a7a82e-0eee-4a3d-8e22-2a19c69c766f/?IncludeEnvStatus=true&IncludeModules=true

Note: We need the extra flags (IncludeEnvStatus and IncludeModules) to obtain the necessary module version key information.

Response body:

```json
{
  "Key": "c9a7a82e-0eee-4a3d-8e22-2a19c69c766f",
  "Name": "EmployeeBackoffice",
}
```
The "EmployeeBackoffice" application has only one module. We take note of its version key, included in the "ModuleStatusInEnvs" list:

Module version key: **bc354abb-6691-41ee-9ed3-9454747e2d4d**

Note: If the application had several modules, we would need to take note of all their version keys.

4. Create a new application version

Starting from the "template" URL of the API method for creating a new application version:

```
POST /lifetimeapi/rest/v2/
environments/{EnvironmentKey}/applications/{ApplicationKey}/versions/
```

We change the `{EnvironmentKey}` and `{ApplicationKey}` placeholders to the correct values,
defining the full URL for the request:

Request: POST lifetimeapi/rest/v2/environments/f3582e43-43c7-4bb4-8cbb-d9f6cbbc35e/applications/c9a7a82e-0eee-4a3d-8e22-2a19c69c766f/versions/

We insert the following text in the POST request body (using the module version key retrieved in step 3):

```json
{
  "ChangeLog": "Go live 1.0",
  "Version": "1.0",
  "MobileVersions": [],
  "ModuleVersionKeys": [
    "bc354abb-6691-41ee-9ed3-9454747e2d4d"
  ]
}
```

The response contains the key of the created application version in the "ApplicationVersionKey" element when the operation is successful.

Response body:

```json
{
  "ApplicationVersionKey": "b3d2ef6b-e5c5-4c7e-851e-0547f7bd634c"
}
```

_deploy an application_

In this example we will perform the following generic steps using API calls:

1. Get the environment keys of the source and target environments *(optional if you already have this information)*;

2. Get the base application version key of the desired application *(optional if you already have this information)*;

3. Create a new deployment;

4. Get the details of the created deployment, checking for any conflicts arising from the deployment of the application;

5. Start the created deployment;

6. Get the deployment execution status until the execution has finished and the deployment status is a terminal one (i.e. one of "needs_user_intervention", "aborted", "finished_successful", "finished_unsuccessful", "finished_unsuccessful", "erroneous", "failed", "failed_unavailable", "failed_unavailable_user_intervention", "failed_unavailable_user_intervention").
Check below for example API calls (requests and responses) for each of the presented steps.

1. Get source and target environment keys

We start by calling the API method that returns all the environments available on the infrastructure to determine the source and target environment keys (only necessary when we don't have this information yet):

Request: GET /lifetimeapi/rest/v2/environments/

Response body:

```json
[
  {
    "Key": "f3582e43-43c7-4bb4-8cbb-d9f6cbbc435e",
    "Name": "Development",
    "OSVersion": "11.0.110.0",
    "Order": 0,
    "HostName": "dev-env.company.com",
    "UseHTTPS": true,
    "EnvironmentType": "Development",
    "NumberOfFrontEnds": 1,
    "ApplicationServerType": ".NET",
    "ApplicationServer": "IIS",
    "DatabaseProvider": "SQLServer",
    "IsCloudEnvironment": false
  },
  {
    "Key": "849515f2-b4ff-4aca-a9d6-9407bea655f4",
    "Name": "Production",
    "OSVersion": "11.0.110.0",
    "Order": 1,
    "HostName": "prd-env-1.company.com",
    "UseHTTPS": true,
    "EnvironmentType": "Production",
    "NumberOfFrontEnds": 1,
    "ApplicationServerType": ".NET",
    "ApplicationServer": "IIS",
    "DatabaseProvider": "SQLServer",
    "IsCloudEnvironment": false
  }
]
```

We take note of the environment keys of the Development and Production environments so that we can use them in the next API calls:
2. Get base application version key

Next, we call the API method that returns all the available applications in the infrastructure to find the desired base application version key:

Request: **GET /lifetimeapi/rest/v2/applications/**

Response body:

```json
[  
  {...},  
  {  
    "Key": "c9a7a82e-0eee-4a3d-8e22-2a19c69c766f",
    "Name": "EmployeeBackoffice",
    "Kind": "WebResponsive",
    "Team": "",
    "Description": "",
    "URLPath": "/EmployeeBackoffice",
    "IconHash": "IconHash6a79e71e-c8e5-9e18-115c-cab789517672",
    "IconURL": "/LifeTimeSDK/ApplicationIcon.aspx?ApplicationKey=c9a7a82e-0eee-4a3d-8e22-2a19c69c766f",
    "IsSystem": false,
    "AppStatusInEnvs": [
      
      {  
        "EnvironmentKey": "f3582e43-43c7-4bb4-8cbb-d9f6cbbc035e",
        "BaseApplicationVersionKey": "b3d2ef6b-e5c5-4c7e-851e-0547f7bd634c",
        "IsModified": true,
        "IsModifiedReason": "Module_Modified",
        "IsModifiedMessage": "At least one module was modified since the version 0.1",
        "ConsistencyStatus": "",
        "ConsistencyStatusMessages": "",
        "MobileAppsStatus": [],
        "ModuleStatusInEnvs": [
          
          {  
            "ApplicationKey": "c9a7a82e-0eee-4a3d-8e22-2a19c69c766f",
            "EnvironmentKey": "f3582e43-43c7-4bb4-8cbb-d9f6cbbc035e",
            "ModuleVersionKey": "bc354abb-6691-41ee-9ed3-9454747e2d4d",
            "ConsistencyStatus": "",
            "ConsistencyStatusMessages": ""
          }
        ],
        "DeploymentZoneKey": ""
      }
  ]
],
"DeploymentZoneKey": ""
}]
```
We take note of the base application version key of the "EmployeeBackoffice" in the "Development" environment (i.e. the environment with key "f3582e43-43c7-4bb4-8cbb-d9f6c7bd35e"):

Base Application Version Key: **b3d2ef6b-e5c5-4c7e-851e-0547f7bd634c**

### 3. Create deployment

We proceed by invoking the API method to create a new deployment. We will need to provide the source and target environment keys, as well as the application version key for the application that we want to deploy.

Request: POST /lifetimeapi/rest/v2/deployments/

Request body:

```json
{
    "Notes": "Deployment Created via postman by Operator1",
    "SourceEnvironmentKey": "f3582e43-43c7-4bb4-8cbb-d9f6c7bd35e",
    "TargetEnvironmentKey": "849515f2-b4ff-4aca-a9d6-9407bea655f4",
    "ApplicationOperations": [
        {
            "ApplicationVersionKey": "b3d2ef6b-e5c5-4c7e-851e-0547f7bd634c",
            "DeploymentZoneKey": ""
        }
    ]
}
```

If the operation is successful, the response body will contain the deployment key of the new deployment.

Response body:

```
dce64ad4-2ddf-4e54-a639-3524bcd5b9a1
```

### 4. Get deployment details checking for conflicts

Next we should invoke the API method to get more detail about the created deployment, checking if OutSystems detected any conflicts:
GET /lifetimeapi/rest/v2/deployments/{DeploymentKey}/

After replacing the `{DeploymentKey}` placeholder with the correct key, we get the following final URL:

Request: GET /lifetimeapi/rest/v2/deployments/dce64ad4-2ddf-4e54-a639-3524bcd5b9a1/

Response body:

```
{
    "Deployment": {
        "Key": "dce64ad4-2ddf-4e54-a639-3524bcd5b9a1",
        "SourceEnvironmentKey": "f3582e43-43c7-4bb4-8cbb-d9f6c6bcbb35e",
        "TargetEnvironmentKey": "849515f2-b4ff-4aca-a9d6-9407bfa655f4",
        "Notes": "Deployment Created via postman by Operator1",
        "CreatedOn": "2018-10-10T15:02:09Z",
        "CreatedBy": "John Doe",
        "CreatedByUsername": "sa_op1",
        "SavedOn": "2018-10-10T15:02:10.853Z",
        "SavedBy": "John Doe",
        "SavedByUsername": "sa_op1",
        "StartedOn": "1900-01-01T00:00:00",
        "StartedBy": "",
        "StartedByUsername": "",
        "AbortedOn": "1900-01-01T00:00:00",
        "AbortedBy": "",
        "AbortedByUsername": "",
        "ApplicationOperations": [
            {
                "ApplicationKey": "c9a7a82e-0eee-4a3d-8e22-2a19c69c766f",
                "ApplicationVersionKey": "b3d2ef6b-e5c5-4c7e-851e-0547f7bd634c",
                "DeploymentOperation": "Deploy 1.0",
                "DeploymentZoneKey": ""
            }
        ],
        "ApplicationsToRedeploy": [],
        "ApplicationConflicts": []
    }
}
```

The **ApplicationConflicts** element is empty, which means that no conflicts were detected.

5. Start the created deployment

We can now start the deployment execution by calling the following API method with the "start" command:

https://success.outsystems.com/Documentation/11/Reference/OutSystems_APIs/LifeTime_Deployment_API_v2/LifeTime_De…

Updated: Mon, 18 Nov 2019 00:58:26 GMT
POST /lifetimeapi/rest/v2/deployments/{DeploymentKey}/{Command}/

Adapting the template above to our example:

Request: POST /lifetimeapi/rest/v2/deployments/
dce64ad4-2ddf-4e54-a639-3524bcd5b9a1/start/

If the "start" command is successful, OutSystems will return a 202 Accepted HTTP status code, with an empty response body.

6. Check deployment execution status

At this point, we use the following API method to determine the execution status of the previously issued "start" command:

GET /lifetimeapi/rest/v2/deployments/{DeploymentKey}/status/

Adapting the template above to our example:

GET /lifetimeapi/rest/v2/deployments/
dce64ad4-2ddf-4e54-a639-3524bcd5b9a1/status/

Response body:

```json
{
  "DeploymentStatus": "finished_successful",
  "Info": "",
  "DeploymentLog": [
    {
      "Instant": "2018-10-10T15:36:11.797Z",
      "Message": "Uploading modules from Development to Production"
    },
    {
      "Instant": "2018-10-10T15:36:14.887Z",
      "Message": "Uploading 'EmployeeBackoffice' from Development to Production"
    },
    {
      "Instant": "2018-10-10T15:36:18.337Z",
      "Message": "Applying Deployment Zones configurations in Production Environment"
    },
    {
      "Instant": "2018-10-10T15:36:21.433Z",
      "Message": "Creating solution pack in Production"
    },
    {...}
  ]
}
```


Updated: Mon, 18 Nov 2019 00:58:26 GMT
Since the deployment status is "finished_successful", we know that the application deployment has finished successfully.

If we get an intermediate deployment status like "running" in the response, we would need to keep polling the status of the deployment execution (i.e. by repeating the API method call) until the execution reached a terminal deployment status, either a successful or an unsuccessful one.