

Hands-on Lab: Base Acronis Cyber Platform API operations with PowerShell

- [Hands-on Lab: Base Acronis Cyber Platform API operations with PowerShell](#)
 - [Hands-on Lab Code Directory](#)
 - [The Acronis Cyber Platform API general workflow](#)
 - [Prerequisites and basis information](#)
 - [Exercise 1: Create an API Client to access the API](#)
 - [Implementation details](#)
 - [Step-by-step execution and checks](#)
 - [Exercise 2: Issue a token to access the API](#)
 - [Implementation details](#)
 - [Step-by-step execution and checks](#)
 - [Exercise 3: Create partner, customer and user tenants and set offering items](#)
 - [Implementation details](#)
 - [Step-by-step execution and checks](#)
 - [Exercise 4: Get a tenant usage](#)
 - [Implementation details](#)
 - [Step-by-step execution and checks](#)
 - [Exercise 5: Create and download simple report](#)
 - [Implementation details](#)
 - [Step-by-step execution and checks](#)
 - [Exercise 6: Add marks to your API calls for better support](#)
 - [Implementation details](#)
 - [Step-by-step execution and checks](#)
 - [Summary](#)

Hands-on Lab Code Directory

File name	File description
<code>0-basis-configuration.ps1</code>	Initialize global variables <code>\$baseUrl</code> , <code>\$partnerTenant</code> , <code>\$customerTenant</code> and <code>\$edition</code> from config files <code>cyber.platform.cfg.json</code> and <code>cyber.platform.cfg.defaults.json</code> .
<code>0-basis-api-check.ps1</code>	Base sanity checks need to be performed before the API calls.
<code>0-basis-functions.ps1</code>	Contains some utilities functions to simplify the API usage.

File name	File description
1-create_client_to_access_api.ps1	Creates an API Client (<code>client_id</code> , <code>client_secret</code>) to generate a JWT token and access the API. The Basic Authentication is used. For Acronis Cyber Protect (Acronis Cyber Cloud 9.0) the Management Console can be used to create an API Client. The result of the script is stored in clear text <code>api_client.json</code> file. It's raw answer from the API call. For your solutions, please, implement secured storage for <code>client_id</code> , <code>client_secret</code> as they are credentials to access the API. The scrip asks for login and password to create an API Client.
2-issue_token.ps1	Issue a JWT token to access the API. The token is expired in 2 hours. During the sanity checks in <code>0-basis-api-check.ps1</code> an expiration time for the current token is checked and a token is reissued if needed. The result of the script is stored in clear text <code>api_token.json</code> file. It's raw answer from the API call. For your solutions, please, implement secured storage for a JWT token info as they are credentials to access the API.
3-0-create_partner_tenant.ps1	Creates a partner with name <u>MyFirstPartner</u> and enables all available offering items dor them for an edition, specified in json configuration files <code>cyber.platform.cfg.json</code> and <code>cyber.platform.cfg.defaults.json</code> .
3-1-create_customer_tenant.ps1	Creates a customer for <u>MyFirstPartner</u> with name <u>MyFirstCustomer</u> and enables all available offering items dor them for an edition, specified in json configuration files <code>cyber.platform.cfg.json</code> and <code>cyber.platform.cfg.defaults.json</code> .
3-2-create_user_activate.ps1	Creates a user for <u>MyFirstCustomer</u> and activate them by setting a password. The script asks for username to create.
4-get_tenant_usages.ps1	Gets usage for the root tenant.
5-create_and_download_simple_report.ps1	Create an one time report to dave for the root tenant, wait till its creation and download.
LICENSE	The license for the code. It's MIT license.
README.md	This file.

File name	File description
cyber.platform.cfg.defaults.json	Contains default configuration values for the scripts. They are used when the values are not defined in cyber.platform.cfg.json file.
cyber.platform.cfg.json	Contains configuration values for the scripts.

The Acronis Cyber Platform API general workflow

#	Operation	When/Period	Prerequisites / Inputs
1	Create an API client under which an integration will be authorized	Initially. Periodically if security policies require your company to regenerate all passwords each X months. Through the API or the Management Portal for ACC 9.0 and greater.	Login and password with a needed level of access in Acronis Cyber Cloud. Usually, it's a service Admin account under your company's Partner tenant in Acronis Cyber Cloud.
2	Issue an access token	1. Before the first API Call which is not connected to the authorization flow 2. Each time when your token is near to be expired.	Your API Client credentials
3	Make API calls		An access token issued using your API Client credentials

Prerequisites and basis information

To run the scripts, you need to edit or create the `cyber.platform.cfg.json` file to provide base parameters. At minimum you need to change `base_url` to your data center URL. The global variables `$baseUrl` initialized from the config file and used for all API requests. All other values can remain unchanged. A `cyber.platform.cfg.json` file example:

```
{
  "base_url": "https://dev-cloud.acronis.com/",
  "partner_tenant": "partner",
  "customer_tenant": "customer",
  "edition": "standard"
}
```

Exercise 1: Create an API Client to access the API

Implementation details

A JWT token with a limited time to life approach is used to securely manage access of any API clients, like our scripts, for the Acronis Cyber Cloud. Using a login and password for a specific user is not a secure and manageable way to create a token, but technically it's possible. Thus, we create an API client with a client id and a client secret to use as credentials to issue a JWT token. To create an API Client, we call the `/clients` end-point with POST request specifying in the JSON body of the request a tenant we want to have access to. To authorize this the request, the Basic Authorization with user login and password for Acronis Cyber Cloud is used.



In Acronis Cyber Cloud 9.0 API Client credentials can be generated in the Management Portal.



Creating an API Client is a one-time process. As the API client is used to access the API, treat it as credentials and store securely. Also, do not store the login and password in the scripts itself.

In the following code block a login and a password are requested from a command line and use it for a Basic Authorization for following HTTP requests.

```
# Get credentials from command line input
$cred = (Get-Credential).GetNetworkCredential()

# Use Login and Password to create an API client
$login = $cred.UserName
$password = $cred.Password
```

In those scripts it is expected that the [Acronis Developer Sandbox](#) is used. It is available for registered developers at [Acronis Developer Network Portal](#). So the base URL for all requests (<https://devcloud.acronis.com/>) is used. Please, replace it with correct URL for your production environment if needed. For more details, please, review the [Authenticating to the platform via the Python shell tutorial](#) from the Acronis Cyber Platform documentation.

For demo purposes, this script issues an API client for a tenant for a user for whom a login and a password are specified. You should add your logic as to what tenant should be used for the API Client creation.

```
# Get Self information to have tenant_id
$myInfo = Invoke-RestMethod -Uri "${baseUrl}api/2/users/me" -Headers $headers
$tenantId = $myInfo.tenant_id

# Body JSON, to request an API Client for the $tenantId
$json = @"
{
  "type": "agent",
  "tenant_id": "$tenantId",
  "token_endpoint_auth_method": "client_secret_basic",
  "data": {
```

```
"client_name": "PowerShell.App"
}
}
"@
```



client_name value defines the name you will see in the ACC 9.0 Management Console. For real integrations, please, name it carefully to have a way to identify it in a future.

```
# Create an API Client
$client = Invoke-RestMethod -Method Post -Uri "${baseUrl}api/2/clients" -Headers $hea

# Save the API Client info to file for further usage
# YOU MUST STORE YOUR CREDENTIALS IN SECURE PLACE
# A FILE USES FOR CODE SIMPLICITY
$client | ConvertTo-Json -Depth 100 | Out-File "api_client.json"
```



A generated client is inherited access rights from a user used for the generation but it's disconnected from them. You don't need to issue a new client even if the user account is removed from Acronis Cloud.



Treat API Clients as a specific service account with access to your cloud. All internal security policies applied to your normal account operations should be in place for API Clients. Thus, don't create new API Clients if you don't really required and disable/delete unused API Clients through the Management Console or API Calls.



You can receive a client_secret only once, just at the issue time. If you loose your client_secret further you must reset secret for the client through the Management Console or API Calls. Please, be aware, that all the tokens will be invalidated.



You need to securely store the received credentials. For simplicity of the demo code, a simple JSON format is used for api_client.json file. Please remember to implement secure storage for your client credentials.

Step-by-step execution and checks

1. Open any available PowerShell environment: Linux, Mac or Windows.
2. Copy code directory to your local system and ensure that all .ps1 files are executable in Linux and Mac cases. We will use Windows PowerShell for this instructions. Your directory listing should looks like bellow.

Mode	LastWriteTime		Length	Name
----	-----	-----	-----	----
-a----	12.03.2020	11:08	84	.gitignore
-a----	12.03.2020	11:08	829	0-basis-api-check.ps1
-a----	12.03.2020	11:08	1286	0-basis-configuration.ps1
-a----	12.03.2020	11:08	3713	0-basis-functions.ps1
-a----	12.03.2020	11:08	1718	1-create_client_to_access_api.ps1
-a----	12.03.2020	11:08	1503	2-issue_token.ps1
-a----	12.03.2020	11:08	1350	3-0-create_partner_tenant.ps1
-a----	12.03.2020	11:08	1578	3-1-create_customer_tenant.ps1
-a----	12.03.2020	11:08	1887	3-2-create_user_activate.ps1
-a----	12.03.2020	11:08	1050	4-get_tenant_usages.ps1
-a----	12.03.2020	11:08	2236	5-create_and_download_simple_report.ps1
-a----	12.03.2020	11:08	138	cyber.platform.cfg.defaults.json
-a----	12.03.2020	11:08	138	cyber.platform.cfg.json
-a----	12.03.2020	11:08	1107	LICENSE
-a----	12.03.2020	11:08	22564	README.md

3. Edit `cyber.platform.cfg.json` file to enter your `base_url` aka your data center URL for API calls. All other options remain unchanged.
4. Type 1 and press Tab, it should autocomplete to the `.\1-create_client_to_access_api.ps1`.
5. Press Enter. You should see a credentials request window.

Enter your username

and password and press OK.

6. If you enter login and password correctly, the script just makes a series of API calls silently and exit. If you make a mistake, you receive a detailed error description. For example, below an error you receive when your login or/and password are incorrect.

```
PS C:\Users\Stanislav.Pavlov\Repos\pwwh_hol> .\1-create_client_to_access_api.ps1

cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:
Credential
Invoke-RestMethod : {"error":{"code":1000,"message":"An error has occurred.", "details":{"info":"","addition":{({}}), "context":{(), "domain":"PlatformAccountServer"}}}
At C:\Users\Stanislav.Pavlov\Repos\pwwh_hol\1-create_client_to_access_api.ps1:26 char:11
+ $myInfo = Invoke-RestMethod -Uri "${baseurl}api/2/users/me" -Headers ...
+ ~~~~~
+ CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-RestMethod], WebException
+ FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeRestMethodCommand
Invoke-RestMethod : {"error":{"code":1000,"message":"An error has occurred.", "details":{"info":"","addition":{({}}), "context":{(), "domain":"PlatformAccountServer"}}}
At C:\Users\Stanislav.Pavlov\Repos\pwwh_hol\1-create_client_to_access_api.ps1:42 char:11
+ $client = Invoke-RestMethod -Method Post -Uri "${baseurl}api/2/client ...
+ ~~~~~
+ CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-RestMethod], WebException
+ FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeRestMethodCommand
```

7. Type `.\api_client.json` and press Enter. You should see the JSON file is opened in your default JSON editor with an API Client information. In this tutorial, we use Visual Studio Code as the default editor. If you can see something similar to picture bellow, you successfully created an API Client and can follow to the next exercise.

```

1  {
2    "registration_access_token": "1770cb5a612c4a9ea44663d8df0f57a5",
3    "client_id": "25fc6e0d-8cbb-4fb8-8d89-a6e0817cc727",
4    "tenant_id": "53b471e1-50dd-4557-ae6d-f687946821d4",
5    "token_endpoint_auth_method": "client_secret_basic",
6    "created_by": "bfa9d672-35ee-4cad-b3d9-024d145f7fe3",
7    "client_secret_expires_at": 0,
8    "type": "agent",
9    "redirect_uris": [
10
11      ],
12    "created_at": "2020-03-12T08:17:09+00:00",
13    "data": {
14      "client_name": "PowerShell.App"
15    },
16    "status": "enabled",
17    "client_secret": "2ce8dfd5a9d3413eab4065e98e5038e2"
18  }
19

```

Exercise 2: Issue a token to access the API

Implementation details

A `client_id` and a `client_secret` can be used to access the API using the Basic Authorization but it's not a secure way as we discussed above. It's more secure to have a JWT token with limited life-time and implement a renew/refresh logic for that token.

To issue a token `/idp/token` end-point is called using POST request with param `grant_type` equal `client_credentials` and content type `application/x-www-form-urlencoded` with Basic Authorization using a `client_id` as a user name and a `client_secret` as a password.

```

# Read an API Client info from a file and store client_id and client_secret in variab
$client = Get-Content "api_client.json" | ConvertFrom-Json
$clientId = $client.client_id
$clientSecret = $client.client_secret

# Manually construct Basic Authentication Header
$pair = "${clientId}:${clientSecret}"
$bytes = [System.Text.Encoding]::ASCII.GetBytes($pair)
$base64 = [System.Convert]::ToBase64String($bytes)
$basicAuthValue = "Basic $base64"
$headers = @{ "Authorization" = $basicAuthValue }

# Use param to tell type of credentials we request
$postParams = @{ grant_type = "client_credentials" }

# Add the request content type to the headers

```



```
$headers.Add("Content-Type", "application/x-www-form-urlencoded")

$token = Invoke-RestMethod -Method Post -Uri "${baseUrl}api/2/idp/token" -Headers $he

# Save the Token info to file for further usage
# YOU MUST STORE YOUR CREDENTIALS IN SECURE PLACE
# A FILE USES FOR CODE SIMPLICITY
# PLEASE CHECK TOKEN VALIDITY AND REFRESH IT IF NEEDED
$token | ConvertTo-Json -Depth 100 | Out-File "api_token.json"
```



You need to securely store the received token. For simplicity of the demo code, the received JSON format is used api_token.json file. Please implement secure storage for your tokens.



A token has time-to-live and must be renewed/refreshed before expiration time. The best practice is to check before starting any API calls sequence and renew/refresh if needed.



Currently, the default time-to-live to a token for the API is 2 hours.

Assuming that the token is stored in the JSON response format as above, it can be done using the following functions set.

expires_on is a time when the token will expire in Unix time format -- seconds from January 1, 1970. Here we assume that we will renew/refresh a token 15 minutes before the expiration time.

```
# Check if the token valid at least 15 minutes
function Confirm-Token {

    [CmdletBinding()]
    Param(
    )

    # Read an token info from
    $token = Get-Content "api_token.json" | ConvertFrom-Json

    $unixTime = $token.expires_on

    $expireOnTime = Convert-FromUnixDate -UnixTime $unixTime
    $timeDifference = New-TimeSpan -End $expireOnTime

    $timeDifference.TotalMinutes -gt 15
}

function Convert-FromUnixDate {

    [CmdletBinding()]
    Param(
        [parameter(Mandatory = $true)]
    )
}
```



```

        [int]
        $UnixTime
    )

    [timezone]::CurrentTimeZone.ToLocalTime(([datetime]'1/1/1970').AddSeconds($UnixTime)
}

function Update-Token {

    [CmdletBinding()]
    Param(
        [parameter(Mandatory = $true)]
        [string]
        $BaseUrl
    )

    # Read an API Client info from a file and store client_id and client_secret in var
    $client = Get-Content "api_client.json" | ConvertFrom-Json
    $clientId = $client.client_id
    $clientSecret = $client.client_secret

    # Manually construct Basic Authentication Header
    $pair = "${clientId}:${clientSecret}"
    $bytes = [System.Text.Encoding]::ASCII.GetBytes($pair)
    $base64 = [System.Convert]::ToBase64String($bytes)
    $basicAuthValue = "Basic $base64"
    $headers = @{ "Authorization" = $basicAuthValue }

    # Use param to tell type of credentials we request
    $postParams = @{ grant_type = "client_credentials" }

    # Add the request content type to the headers
    $headers.Add("Content-Type", "application/x-www-form-urlencoded")

    $token = Invoke-RestMethod -Method Post -Uri "${BaseUrl}api/2/idp/token" -Headers $
    $headers

    # Save the Token info to file for further usage
    # YOU MUST STORE YOUR CREDENTIALS IN SECURE PLACE
    # A FILE USES FOR CODE SIMPLICITY
    # PLEASE CHECK TOKEN VALIDITY AND REFRESH IT IF NEEDED
    $token | ConvertTo-Json -Depth 100 | Out-File "api_token.json"

    $token.access_token
}

```

Step-by-step execution and checks

1. Type 2 and press Tab, it should autocomplete to the .\2-issue_token.ps1.
2. Press Enter. If api_client.json file exists and contains correct information, the script just makes a series of API calls silently and exit. If you make a mistake, you receive a detailed error description.
3. Type .\api_token.json and press Enter. You should see the JSON file with a token information opened in your default editor. If you can see something similar to picture bellow, you successfully issued a token and can follow to the next exercise.

```

1 {
2   .... "id_token": "...eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjUzZjNkZGY4LWYyOTQwMDEyLTg5MDA...
3   .... "access_token": "...eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjUzZjNkZGY4LWYyOTQwMDEyLTg5MDA...
4   .... "expires_on": ...1584009425,
5   .... "token_type": ... "bearer"
6 }
7

```

4. Including `0-basis-api-check.ps1` file in each following scripts we ensure that a token will be reissued if needed before any API call.
5. Check `0-basis-api-check.ps1` file to verify that you can understand implementation details described above.

Exercise 3: Create partner, customer and user tenants and set offering items

Implementation details

So now we can securely access the Acronis Cyber Platform API calls. In this topic we discuss how to create a partner, a customer tenants and enable for them all available offering items, and then create a user for the customer and activate the user by setting a password.

As we discussed above, before making a call to the actual API you need to ensure that an authorization token is valid. Please, use the functions like those described above to do it.

Assuming that we create the API client for our root tenant, we start from retrieving the API Client tenant information using GET request to `/clients/${clientId}` end-point. Then, using received `tenant_id` information as a parameter and `kind` equal to `partner`, we build a JSON body for POST request to `/tenants` end-point to create the partner. Next, we are going to enable all applications and offering items for the tenants. Briefly, we take all available offering items for the parent tenant of the partner or the customer using GET request to `/tenants/${tenantId}/offering_items/available_for_child` end-point with needed query parameters specifying edition and kind of the tenant. Then, we need to enable these offering items for the partner or the customer using PUT request to `/tenants/${tenantId}/offering_items` end-point with all offering items JSON in the request body and appropriate `tenantId`.



The following kind values are supported root, partner, folder, customer, unit.

```
# Get Root tenant_id for the API Client
$client = Get-Content "api_client.json" | ConvertFrom-Json
$clientId = $client.client_id

$apiClientInfo = Invoke-RestMethod -Uri "${baseUrl}api/2/clients/${clientId}" -Headers $headers
$tenantId = $apiClientInfo.tenant_id

# Body JSON, to create a partner tenant
$json = @"
{
    "name": "MyFirstPartner",
    "parent id": "${tenantId}",

```

```

        "kind": "${partnerTenant}"
    }
"@

# Create a partner
$partner = Invoke-RestMethod -Method Post -Uri "${baseUrl}api/2/tenants" -Headers $headers
$partnerId = $partner.id

Enable-AllOfferingItems -BaseUrl $baseUrl -ParentTenantID $tenantId -TenantID $partnerId

# Save the JSON partner info into a file
$partner | ConvertTo-Json -Depth 100 | Out-File "partner.json"

```

This is absolutely the same process as for a customer, the only difference is kind equal to customer in the request body JSON and /offering_items/available_for_child parameters.

```

# Get a partner info
$partner = Get-Content "partner.json" | ConvertFrom-Json
$partnerId = $partner.id

# Body JSON, to create a customer tenant
$json = @"
{
    "name": "MyCustomer",
    "parent_id": "${partnerId}",
    "kind": "${customerTenant}"
}
"@


# Create a customer in a trial mode
$customer = Invoke-RestMethod -Method Post -Uri "${baseUrl}api/2/tenants" -Headers $headers
$customerId = $customer.id

# Save the JSON customer info into a file
$customer | ConvertTo-Json -Depth 100 | Out-File "customer.json"

Enable-AllOfferingItems -BaseUrl $baseUrl -ParentTenantID $partnerId -TenantID $customerId

```

By default, customers are created in a trial mode. To switch to production mode we need to update customer pricing. To perform this task, we start from requesting current pricing using a GET request to /tenants/\${customerTenantId}/pricing end-point then change mode property to production in the received JSON, then, finally, update the pricing using PUT request to /tenants/\${customerTenantId}/pricing end-point with a new pricing JSON.

 **Please, be aware, that this switch is non-revertible.**

```

# Switching customer tenant to production mode
$customerPricing = Invoke-RestMethod -Uri "${baseUrl}api/2/tenants/${customerId}/pricing"
$customerPricing.mode = "production"

```

```
$customerPricingJson = $customerPricing | ConvertTo-Json

Invoke-RestMethod -Method Put -Uri "${baseUrl}api/2/tenants/${customerId}/pricing" -H
```

Finally, we create a user for the customer. At first, we check if a login is available using GET request to /users/check_login end-point with username parameter set to an expected login. Then, we create a JSON body for POST request to /users end-point to create a new user.

```
# Get a customer info
$customer = Get-Content "customer.json" | ConvertFrom-Json
$customerId = $customer.id

$userLogin = Read-Host -Prompt "Enter expected user login:"
$userLoginParam = @{username = $userLogin }

$response = Invoke-WebRequest -Uri "${baseUrl}api/2/users/check_login" -Headers $hea

# Check if Login name is free
if ($response.StatusCode -eq 204) {

# Body JSON, to create a user
$json = @"
{
  "tenant_id": "${customerId}",
  "login": "${userLogin}",
  "contact": {
    "email": "${userLogin}@example.com",
    "firstname": "Firstname",
    "lastname": "Lastname"
  }
}
"@

$user = Invoke-RestMethod -Method Post -Uri "${baseUrl}api/2/users" -Headers $heade
$userId = $user.id

# Save the JSON user info into a file
$user | ConvertTo-Json -Depth 100 | Out-File "user.json"
}
```

A created user is not active. To activate them we can either send them an activation e-mail or set them a password. The sending of an activation e-mail is the preferable way, as in this case a user can set their own password by themselves. We use a set password way for demo purposes and a fake e-mail is used. To set a password we send a simple JSON and POST request to /users/\${userId}/password end-point.

```
# Body JSON, to assign a password and activate the user
# NEVER STORE A PASSWORD IN PLAIN TEXT FILE
# THIS CODE IS FOR API DEMO PURPOSES ONLY
# AS IT USES FAKE E-MAIL AND ACTIVATION E-MAIL CAN'T BE SENT
$json = @"
{
  "password": "MyStrongP@ssw0rd"
}
```

```
}
"@
```

```
Invoke-RestMethod -Method Post -Uri "${baseUrl}api/2/users/${userId}/password" -Header
```

At this point, we've created a partner, a customer, enable offering items for them, create a user and activate them.

Step-by-step execution and checks

Create partner and enable all available standard edition offering items

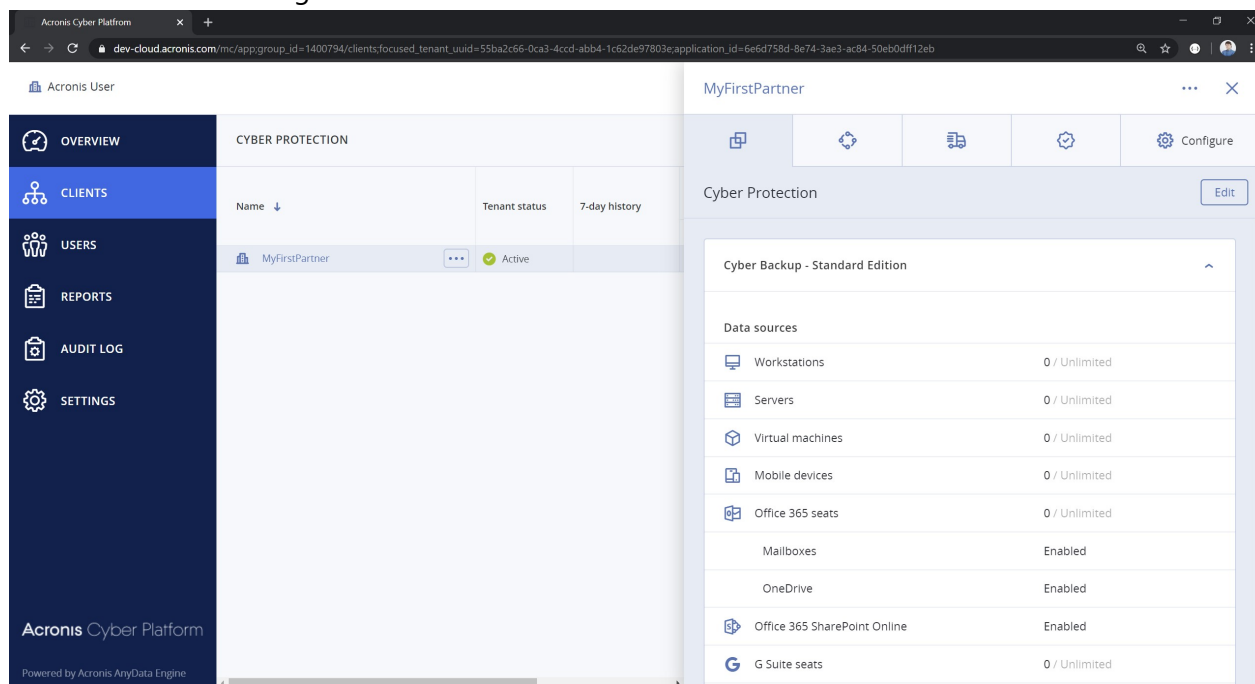
1. Type 3-0 and press Tab, it should autocomplete to the `.\3-0-create_partner_tenant.ps1`.
2. Press Enter. If `api_client.json` file exists and contains correct information, the script just makes a series of API calls, display list of offering items set and exit. If you make a mistake, you receive a detailed error description.

```
PS C:\Users\Stanislav.Pavlov\Repos\pwsh_hol> .\3-0-create_partner_tenant.ps1
items
-----
{@{locked=False; usage_name=compute_points; name=compute_points; infra_id=debe7865-fa8d-4c16-8e26-adcf8d7fd23d; type=infra; edition=standard; status=1; application_id=...
```

3. Type `.\partner.json` and press Enter. You should see the JSON file with a partner information opened in your default editor. If you can see something similar to picture bellow, you successfully created a partner.

```
1  {
2    "update_lock": {
3      "enabled": false,
4      "owner_id": null
5    },
6    "brand_uuid": "d81d89b7-6f63-43ea-ba83-3574e184d0c1",
7    "language": "en",
8    "brand_id": 6194,
9    "version": 1,
10   "name": "MyFirstPartner",
11   "parent_id": "53b471e1-50dd-4557-ae6d-f687946821d4",
12   "contact": {
13     "country": null,
14     "firstname": "",
15     "phone": null,
16     "address1": null,
17     "email": "",
18     "city": null,
19     "lastname": "",
20     "zipcode": null,
21     "address2": null,
22     "state": null
23   },
24   "customer_id": null,
25   "has_children": false,
26   "ancestral_access": true,
27   "kind": "partner",
28   "internal_tag": null,
29   "id": "551a2660-8041-4163-b302-070000000000"
```

- Open the Management Portal and check that a new partner with name MyFirstPartner was created and for them all offering items for standard edition were enabled.



Create customer, enable all available standard edition offering items and switch to production mode

- Type 3-1 and press Tab, it should autocomplete to the `.\3-1-create_customer_tenant.ps1`.
- Press Enter. If `api_client.json` file exists and contains correct information, the script just makes a series of API calls, display list of offering items set and exit. If you make a mistake, you receive a detailed error description.

```
PS C:\Users\Stanislav.Pavlov\Repos\psh_hol> .\3-0-create_partner_tenant.ps1
items
-----
{@"locked=False; usage_name=compute_points; name=compute_points; infra_id=debe7865-fa8d-4c16-8e26-adcf8d7fd23d; type=infra; edition=standard; status=1; application_id..."}
```

- Type `.\customer.json` and press Enter. You should see highlighted JSON file with a customer information. If you can see something similar to picture bellow, you successfully created a customer.


```

1  {
2    "update_lock": {
3      "enabled": false,
4      "owner_id": null
5    },
6    "brand_uuid": "d81d89b7-6f63-43ea-ba83-3574e184d0c1",
7    "language": "en",
8    "brand_id": 6194,
9    "version": 1,
10   "name": "MyCustomer",
11   "parent_id": "55ba2c66-0ca3-4ccd-abb4-1c62de97803e",
12   "contact": {
13     "country": null,
14     "firstname": "",
15     "phone": null,
16     "address1": null,
17     "email": "",
18     "city": null,
19     "lastname": "",
20     "zipcode": null,
21     "address2": null,
22     "state": null
23   },
24   "customer_id": null,
25   "has_children": false,
26   "ancestral_access": true,
27   "kind": "customer",
28   "internal_tag": null,
29   "id": "cd2a5fd4-5dbd-49b2-8300-5b99a7e740a9",

```

4. Open the Management Portal and check that a new customer with name MyFirstCustomer was created under MyFirstPartner and for them all offering items for standard edition were enabled.

The screenshot shows the Acronis Cyber Platform Management Portal. On the left, a sidebar contains navigation links: OVERVIEW, CLIENTS, USERS, REPORTS, AUDIT LOG, and SETTINGS. The main area displays a table of customers under the 'MyFirstPartner' tenant. The table has columns for Name, Tenant status, and 7-day history. A single customer, 'MyCustomer', is listed with a status of 'Active' and 'No Data' in the history column. To the right, a detailed view of 'MyCustomer' is shown, including a 'Cyber Protection' section with a table of enabled services:

Cyber Backup - Standard Edition	
Data sources	
Workstations	0 / Unlimited
Servers	0 / Unlimited
Virtual machines	0 / Unlimited
Mobile devices	0 / Unlimited
Office 365 seats	0 / Unlimited
Mailboxes	Enabled
OneDrive	Enabled
Office 365 SharePoint Online	Enabled
G Suite seats	0 / Unlimited

Create user, activate them by setting a password and enable backup services

1. Type 3-2 and press Tab, it should autocomplete to the `.\3-2-create_user_activate.ps1`.
2. Press Enter. You should see request for expected username. Type it and press Enter.

```
PS C:\Users\Stanislav.Pavlov\Repos\pwsh_hol> .\3-2-create_user_activate.ps1
Enter expected username: stasX
```

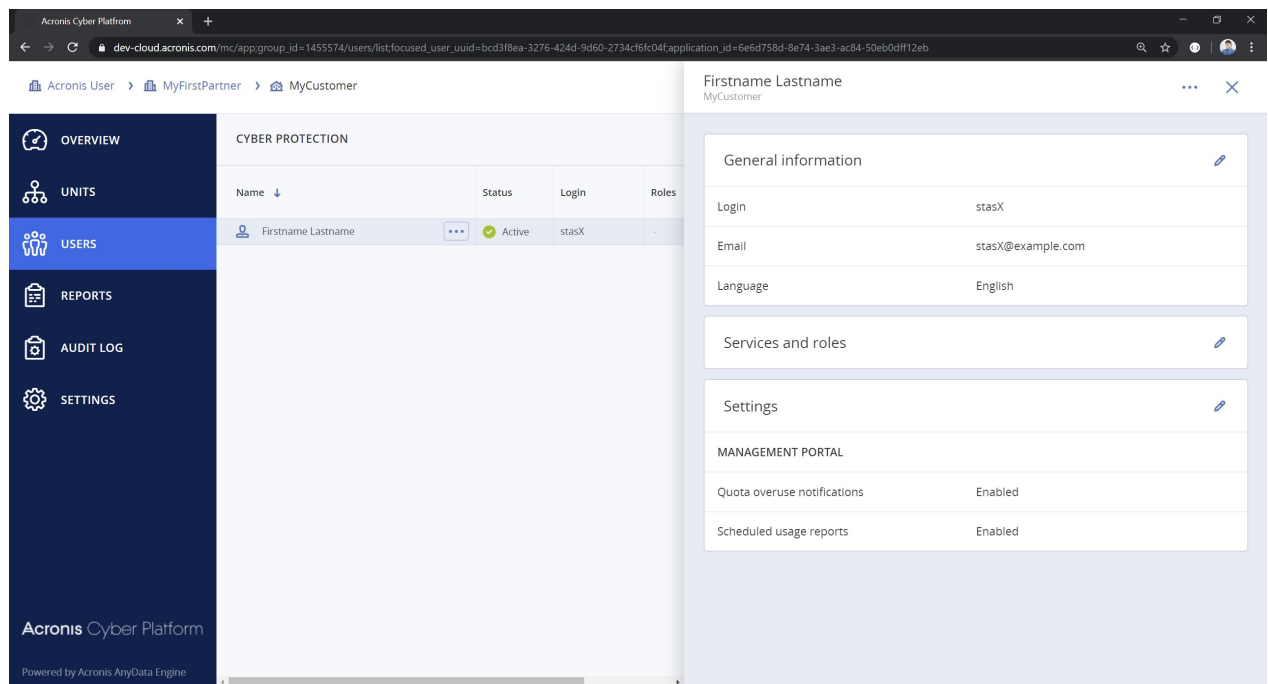
3. If `api_client.json` file exists and contains correct information, and a user with this username doesn't exist, the script just makes a series of API calls silently and exit. If a user with provided username exists or any other issue exists, you receive a detailed error description.


```
PS C:\Users\Stanislav.Pavlov\Repos\pwsh_hol> .\3-2-create_user_activate.ps1
Enter expected username: stasX
Invoke-WebRequest : {"error":{"code":409,"message":"User 'stasX' exists","details":{"info":null,"addition":{},"context":{},"domain":"PlatformAccountServer"}}}
At C:\Users\Stanislav.Pavlov\Repos\pwsh_hol\3-2-create_user_activate.ps1:21 char:13
+ $response = Invoke-WebRequest -Uri "${baseUrl}api/2/users/check_login ...
+ ~~~~~
+ CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
+ FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
Can't create a new user. A user with this login already exists.
```

4. Type `.\user.json` and press Enter. You should see the JSON file with a user information opened in your default editor. If you can see something similar to picture below, you successfully created and activated a user.


```
1  {
2      "language": "en",
3      "contact": {
4          "country": null,
5          "firstname": "Firstname",
6          "phone": null,
7          "address1": null,
8          "email": "stasX@example.com",
9          "city": null,
10         "lastname": "Lastname",
11         "zipcode": null,
12         "address2": null,
13         "state": null
14     },
15     "version": 1,
16     "tenant_id": "cd2a5fd4-5dbd-49b2-8300-5b99a7e740a9",
17     "idp_id": "11111111-1111-1111-1111-111111111111",
18     "personal_tenant_id": "8e52bff2-fb6a-47ed-a703-b2ec8f593eb5",
19     "created_at": "2020-03-12T09:02:57.022736+00:00",
20     "notifications": [
21         "quota",
22         "reports",
23         "backup_daily_report"
24     ],
25     "login": "stasX",
26     "activated": false,
27     "id": "bcd3f8ea-3276-424d-9d60-2734cf6fc04f",
28     "mfa_status": "disabled",
29     "business_types": [
```


5. Open the Management Portal and check that a new user with provided username was created under MyFirstCustomer and it's in an active state.



 **The created user has no roles assigned. It means it can't use any service. To enable services/applications you need to assign an appropriate role to a user. In next steps you will create a bash script to assign the created user backup_user role to enable backup services.**

6. Copy 3-2-create_user_activate.ps1 file to 6-assign-user-backup-role.ps1 using following command `copy 3-2-create_user_activate.ps1 6-assign-user-backup-role.ps1`.

 **All operations with the user account roles are located under the `/users/{user_id}/access_policies` endpoint.**

 **To build a JSON to assign a role for a user id and user personal_tenant_id need to be known. All these values can be retrieved from the user .json file we've received as result of the user creation API call.**

7. In your preferred editor, open and edit the 6-assign-user-backup-role.ps1. In our following instructions Visual Studio Code editor is used. To open the file in Visual Studio Code editor, type code `.\6-assign-user-backup-role.ps1` and press Enter.

```

> 6-assign-user-backup-role.ps1 X
C: > Users > Stanislav.Pavlov > Repos > pwsh_hol > > 6-assign-user-backup-role.ps1 > ...
1  #*****
2  # Copyright © 2019-2020 Acronis International GmbH. This source code is distributed under MIT software license.
3  #*****
4
5  # include common functions
6  . ".\0-basis-functions.ps1"
7
8  # include base configuration
9  . ".\0-basis-configuration.ps1"
10
11 # include basis API checks
12 . ".\0-basis-api-check.ps1"
13
14 # Get a customer info
15 $customer = Get-Content "customer.json" | ConvertFrom-Json
16 $customerId = $customer.id
17
18 $userLogin = Read-Host -Prompt "Enter expected username"
19 $userLoginParam = @{username = $userLogin}
20
21 $response = Invoke-WebRequest -Uri "${baseUrl}api/2/users/check_login" -Headers $headers -Body $userLoginParam
22
23 # Check if login name is free
24 if ($response.StatusCode -eq 204) {
25
26 # Body JSON, to create a user
27 $json = @"
28 {
29   "tenant_id": "${customerId}",

```

8. Find the following code in the file

```

# Get a customer info
$customer = Get-Content "customer.json" | ConvertFrom-Json
$customerId = $customer.id

```

and edit it to work with user.json

```

# Get a user info
$user = Get-Content "user.json" | ConvertFrom-Json
$userId = $user.id

```

9. Then personal_tenant_id should be retrieved from user.json file. So just add after

```
$userId = $user.id
```

the following code

```
$userPersonalTenantId = $user.personal_tenant_id
```

10. Now all the information to build a JSON body for our request to the API endpoint. Just after the previous \$userPersonalTenantId code, enter the following code

```

$json = @"
{"items": [
  {"id": "00000000-0000-0000-0000-000000000000",
  "issuer_id": "00000000-0000-0000-0000-000000000000",

```

```

        "role_id": "backup_user",
        "tenant_id": "${userPersonalTenantId}",
        "trustee_id": "${userId}",
        "trustee_type": "user",
        "version": 0}
    ]}
"@

```

You can find more information regarding JSON format in the API documentation

<https://developer.acronis.com/doc/platform/management/v2/#/http/models/structures/access-policy>.

11. And finally as all the data ready, let's add code to call the API. To update a user access policy /users/\${userId}/access_policies end-point is called using PUT request with Bearer Authentication and a JSON body.
12. Find the following code in the end of the file and copy it below the JSON

```
Invoke-RestMethod -Method Post -Uri "${baseUrl}api/2/users/${userId}/password" -Head
```

13. Edit this code to make appropriate PUT call

```
Invoke-RestMethod -Method Put -Uri "${baseUrl}api/2/users/${userId}/access_policies"
```

14. Delete all other code below the edited. So finally you should have the following code in the file.

```

#####
# Copyright © 2019-2020 Acronis International GmbH. This source code is distributed u
#####

# include common functions
. ".\0-basis-functions.ps1"

# include base configuration
. ".\0-basis-configuration.ps1"

# include basis API checks
. ".\0-basis-api-check.ps1"

# Get a customer info
$user = Get-Content "user.json" | ConvertFrom-Json
$userId = $user.id
$userPersonalTenantId = $user.personal_tenant_id

$json = @"
{"items": [
    {"id": "00000000-0000-0000-0000-000000000000",
    "issuer_id": "00000000-0000-0000-0000-000000000000",
    "role_id": "backup_user",
    "tenant_id": "${userPersonalTenantId}",
    "trustee_id": "${userId}",
    "trustee_type": "user",
    "version": 0}

```

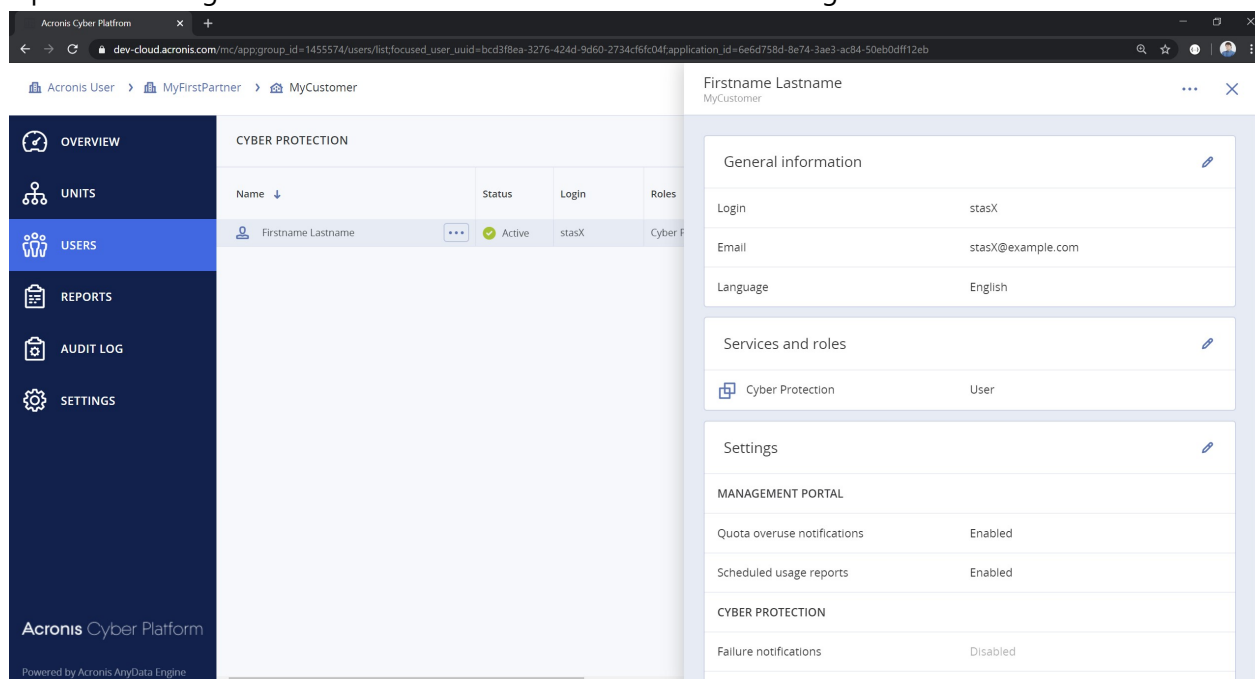
```
}}
"@

Invoke-RestMethod -Method Put -Uri "${baseUrl}api/2/users/${userId}/access_policies"
```

15. Save it. Exit the editor. Type 6 and press Tab, it should autocomplete to the `.\6-assign-user-backup-role.ps1`.
16. Press Enter. If `api_client.json` file exists and contains correct information, the script just makes an API call and return current list of the user access policies and exit. If you make a mistake, you receive a detailed error description.

```
PS C:\Users\Stanislav.Pavlov\Repos\pwsh_ho1> .\6-assign-user-backup-role.ps1
items
-----
{@(trustee_type=user; role_id=backup_user; version=0; tenant_id=8e52bff2-fb6a-47ed-a703-b2ec8f593eb5; trustee_id=bcd3f8ea-3276-424d-9d60-2734cf6fc04f; issuer_id=00000...
```

17. Open the Management Portal and check that the user has the assigned role.



Exercise 4: Get a tenant usage

Implementation details

A very common task is to check a tenant's usage. It's a simple task. We just need to make a GET request to `/tenants/${tenantId}/usages` end-point, as result we receive a list with current usage information in JSON format.

⚠ The information about a service usage of the tenant, provided by the `/tenants/${tenantId}/usages` endpoint, is updated on average every 5-6 hours.

```
# Get Root tenant_id for the API Client
$client = Get-Content "api_client.json" | ConvertFrom-Json
$clientId = $client.client_id

$apiClientInfo = Invoke-RestMethod -Uri "${baseUrl}api/2/clients/${clientId}" -Header
```

```

$tenantId = $apiClientInfo.tenant_id

# Get Usage List for specific tenant
$itemsList = Invoke-RestMethod -Uri "${baseUrl}api/2/tenants/${tenantId}/usages" -He

# Save JSON usages info into a file
$itemsList | ConvertTo-Json -Depth 100 | Out-File "${tenantId}_usages.json"

```



It's very useful to store usage information for further processing. In our example we use response JSON format to store it in a file.

Step-by-step execution and checks

1. Type 4 and press Tab, it should autocomplete to the `.\4-get_tenant_usages.ps1`.
2. Press Enter. If `api_client.json` file exists and contains correct information, the script just makes a series of API calls silently and exit. If you make a mistake, you receive a detailed error description.
3. Type `dir *_usages.json` and press Enter. You should see the created file name for the usage.

```

PS C:\Users\Stanislav.Pavlov\Repos\pwsh_hol> dir *_usages.json

Directory: C:\Users\Stanislav.Pavlov\Repos\pwsh_hol

Mode                LastWriteTime         Length Name
----                -
-a----           12.03.2020    13:52         176232 53b471e1-50dd-4557-ae6d-f687946821d4_usages.json

```

4. Type the name of file you found at the previous step and press Enter. You should see the JSON file with a usage information opened in your default editor. If you can see something similar to picture bellow, you successfully retrieve the usage.


```

1  {
2      "items": [
3          {
4              "tenant_uuid": "53b471e1-50dd-4557-ae6d-f687946821d4",
5              "tenant_id": 1400794,
6              "type": "infra",
7              "application_id": "6e6d758d-8e74-3ae3-ac84-50eb0dff12eb",
8              "name": "compute_points",
9              "edition": "standard",
10             "usage_name": "compute_points",
11             "range_start": "2020-03-01T00:00:00",
12             "absolute_value": 0,
13             "value": 0,
14             "measurement_unit": "seconds",
15             "offering_item": {
16                 "status": 1,
17                 "quota": {
18                     "value": null,
19                     "overage": null,
20                     "version": 0
21                 }
22             },
23             "infra_id": "debe7865-fa8d-4c16-8e26-adcf8d7fd23d"
24         },
25         {
26             "tenant_uuid": "53b471e1-50dd-4557-ae6d-f687946821d4",
27             "tenant_id": 1400794,
28             "type": "infra",
29             "application_id": "6e6d758d-8e74-3ae3-ac84-50eb0dff12eb",
30             "name": "dre_compute_points",
31             "edition": "disaster_recovery",
32             "usage_name": "compute_points",
33             "range_start": "2020-03-01T00:00:00",
34             "absolute_value": 0,
35             "value": 0,
36             "measurement_unit": "seconds",
37             "offering_item": {
38                 "status": 1,
39                 "quota": {

```

Exercise 5: Create and download simple report

Implementation details

The reporting capability of the Acronis Cyber Cloud gives you advanced capabilities to understand usage. In the following simple example, we create a one-time report in csv format, and then download it. To check other options, please, navigate to the Acronis Cyber Platform [documentation](#).

To create a report to save, we build a body JSON and make a POST request to /reports end-point. Then we look into stored reports with specified \$reportId making a GET request to /reports/\${reportId}/stored endpoint.

```

# Get Root tenant_id for the API Client
$client = Get-Content "api_client.json" | ConvertFrom-Json
$clientId = $client.client_id

$apiClientInfo = Invoke-RestMethod -Uri "${baseUri}api/2/clients/${clientId}" -Header
$tenantId = $apiClientInfo.tenant_id

```



```

# Body JSON to create a report
$json = @"
{
    "parameters": {
        "kind": "usage_current",
        "tenant_id": "$tenantId",
        "level": "accounts",
        "formats": [
            "csv_v2_0"
        ]
    },
    "schedule": {
        "type": "once"
    },
    "result_action": "save"
}
"@

# Create a report
$report = Invoke-RestMethod -Method Post -Uri "${baseUrl}api/2/reports" -Headers $hea

# Save JSON report info into a file
$reportId = $report.id
$report | ConvertTo-Json -Depth 100 | Out-File "${reportId}_report_for_tenant_${tenantId}.json"

# A report is not produced momentarily, so we need to wait for it to become saved
# Here is a simple implementation for sample purpose expecting that
# For sample purposes we use 1 report from stored -- as we use once report
do {
    Start-Sleep -Seconds 1
    # Get the stored report
    $storedReportInfo = Invoke-RestMethod -Uri "${baseUrl}api/2/reports/${reportId}/stored"
} until ($storedReportInfo.items[0].status -eq "saved")

# For sample purposes we use 1 report from stored -- as we use once report
# MUST BE CHANGED if you want to deal with scheduled one or you have multiple reports
$storedReportId = $storedReportInfo.items[0].id

# Download the report
Invoke-WebRequest -Uri "${baseUrl}api/2/reports/${reportId}/stored/${storedReportId}"

```

Step-by-step execution and checks

1. Type 5- and press Tab, it should autocomplete to the .\5-create_and_download_simple_report.ps1.
2. Press Enter. If api_client.json file exists and contains correct information, the script just makes a series of API calls silently and then download report. If you make a mistake, you receive a detailed error description.
3. Type dir *report*.json and press Enter. You should see the created file name for the report.

```

PS C:\Users\Stanislav.Pavlov\Repos\pwsh_hol> dir *report*.json

Directory: C:\Users\Stanislav.Pavlov\Repos\pwsh_hol

Mode                LastWriteTime         Length Name
----
-a-----         12.03.2020    13:59           1716 33bca149-992c-4431-8e69-99966dfd378f_report_for_tenant_53b471e1-50dd-4557-ae6d-f687946821d4.json

```

4. Type the name of file you found at the previous step and press Enter. You should see the JSON file with the crated report information opened in your default editor. If you can see something similar to picture bellow, you successfully created the report.

```

1  {
2    "id": "33bca149-992c-4431-8e69-99966dfd378f",
3    "parameters": {
4      "level": "accounts",
5      "formats": [
6        "csv_v2_0"
7      ],
8      "tenant_id": "53b471e1-50dd-4557-ae6d-f687946821d4",
9      "kind": "usage_current",
10     "period": {
11       "end": "2020-03-11",
12       "start": "2020-03-01"
13     }
14   },
15   "version": 1,
16   "generation_date": "2020-03-12",
17   "result_action": "save",
18   "schedule": {
19     "enabled": true,
20     "type": "once"
21   },
22   "recipients": [
23   ]
24 }
25
26

```

5. Type `dir *_report.csv` and press Enter. You should see the download report file.

```

PS C:\Users\Stanislav.Pavlov\Repos\pwsh_hol> dir *_report.csv

Directory: C:\Users\Stanislav.Pavlov\Repos\pwsh_hol

Mode                LastWriteTime         Length Name
----                -
-a----           12.03.2020   13:59           40543 61e45364-8f32-402d-a8f7-a36bfe1ed881_report.csv

```

6. Use any appropriate editor to open this .csv file.

Exercise 6: Add marks to your API calls for better support

Implementation details

It's technically possible to identify your API calls as they are connected to your API Client. But still it's required a lot of efforts and hard to find in your Audit log at the Management Portal for your. Thus to better support your development effort it would be a great idea to identify your integrations and API calls somehow. Traditional way to do it in a RESTful world is using the User-Agent header.

There are common recommendations how to build your User-Agent header:

```
User-Agent: <product>/<product-version> <comment>
```

For example, for our hands-on lab, you can use:

```
User-Agent: Training/1.0 Acronis #CyberFit Developers Business Automation Training
```

To implement it using our bash examples, we need just add the header to each Invoke-RestMethod call using API:

```
$headers.Add("User-Agent", "Training/1.0 Acronis #CyberFit Developers Business Automa
```



Please, for a real integration, use your real integration name, a specific version and suitable comments to simplify your support.

Step-by-step execution and checks

1. Copy 0-basis-api-check.ps1 file to 0-basis-api-check_with_user_agent.ps1 using following command `copy 0-basis-api-check.ps1 0-basis-api-check_with_user_agent.ps1`.
2. In your preferred editor, open and edit the 0-basis-api-check_with_user_agent.ps1.
3. At the end of the file just find

```
$headers.Add("Content-Type", "application/json")
```

and right after this line insert the following

```
$headers.Add("User-Agent", "Training/1.0 Acronis #CyberFit Developers Business Automa
```

4. Save the file. Exit the editor.
5. Rename 0-basis-api-check.ps1 file to 0-basis-api-check_old.ps1 using following command `ren 0-basis-api-check.ps1 0-basis-api-check_old.ps1`.
6. Rename 0-basis-api-check_with_user_agent.ps1 file to 0-basis-api-check_with.ps1 using following command `ren 0-basis-api-check_with_user_agent.ps1 0-basis-api-check.ps1`.
7. So now, in all the code files except 1-create_client_to_access_api.ps1 and 2-issue_token.ps1, all the API call will executed with specific User-Agent.



We will create an API Client in the next step for demo purposes only. Don't forget to delete it after the exercise.

8. To check how our User-Agent affects an audit log you can see in the Management Portal, let's create a new API Client.
9. In your preferred editor, open and edit the 1-create_client_to_access_api.ps1.
10. Find in the file the following line

```
$headers.Add("Content-Type", "application/json")
```

and right after this line insert the following

```
$headers.Add("User-Agent", "Training/1.0 Acronis #CyberFit Developers Business Automa
```

11. Save the file. Exit the editor. 12. Rename `api_client.json` file to `api_client_old.json` using following command `ren api_client.json api_client_old.json`. We are planning to delete the new API Client, so we need to store our previous one.
12. Type 1 and press Tab, it should autocomplete to the `.\1-create_client_to_access_api.ps1`.
13. Press Enter. You should see request for login. Type it and press Enter. You should see request for password. Type it and press Enter
14. If you enter login and password correctly, the script just makes a series of API calls silently and exit. If you make a mistake, you receive a detailed error description.
15. Login to the Management Portal and check how our request are represented in the Audit log.

The screenshot shows the Acronis Cyber Platform Management Portal. On the left is a navigation menu with options: OVERVIEW, CLIENTS, USERS, REPORTS, AUDIT LOG (selected), and SETTINGS. The main area displays the 'AUDIT LOG' table with columns: Event, Severity, Date, and Owner. The table contains multiple entries for 'Offering item was turned on' and 'User was updated'. A modal window titled 'API client was created' is open on the right, showing the 'JSON' tab with the following details:

```
{
  "user": null,
  "context": {
    "new": {
      "data": {
        "client_name": "PowerShell.App"
      }
    },
    "status": "Enabled",
    "client_id": "eefd0300-33e6-45b7-a835-9e250ab9321c",
    "client_type": "Agent",
    "redirect_uris": [],
    "request_meta": {
      "ip_address": "91.195.22.67",
      "user_agent": "Training/1.0 Acronis #CyberFit Developers Business Automation Training",
      "request_type": "api"
    },
    "owner_tenant_uuid": "53b471e1-50dd-4557-ae6d-f687946821d4",
    "token_endpoint_auth_method": "ClientSecretBasic"
  },
  "old": null,
  "severity": 6,
  "subject": {
    "user": {
      "repr": "Stas Pavlov (stas.pavlov@outlook.com)",
      "created_at": "2019-12-02T12:22:13+00:00",
      "language": "en",

```



Don't forget to move the old client JSON file back and delete the new client if you don't plan to use it further.

Summary

Now you know how to use base operations with the Acronis Cyber Platform API:

1. Create an API Client for the Acronis Cyber Platform API access
2. Issue a token for secure access for the API
3. Establish a simple procedure to renew/refresh the token
4. Create a partner and a customer tenants and enable offering items for them.
5. Create a user for a customer tenant and activate them.
6. Enable services for a user by assigning a role.
7. Receive simple usage information for a tenant.

8. Create and download reports for usage.

Get started today, register on the [Acronis Developer Portal](#) and see the code samples available, you can also review solutions available in the [Acronis Cyber Cloud Solutions Portal](#).



Copyright © 2019-2020 Acronis International GmbH. This is distributed under MIT license.