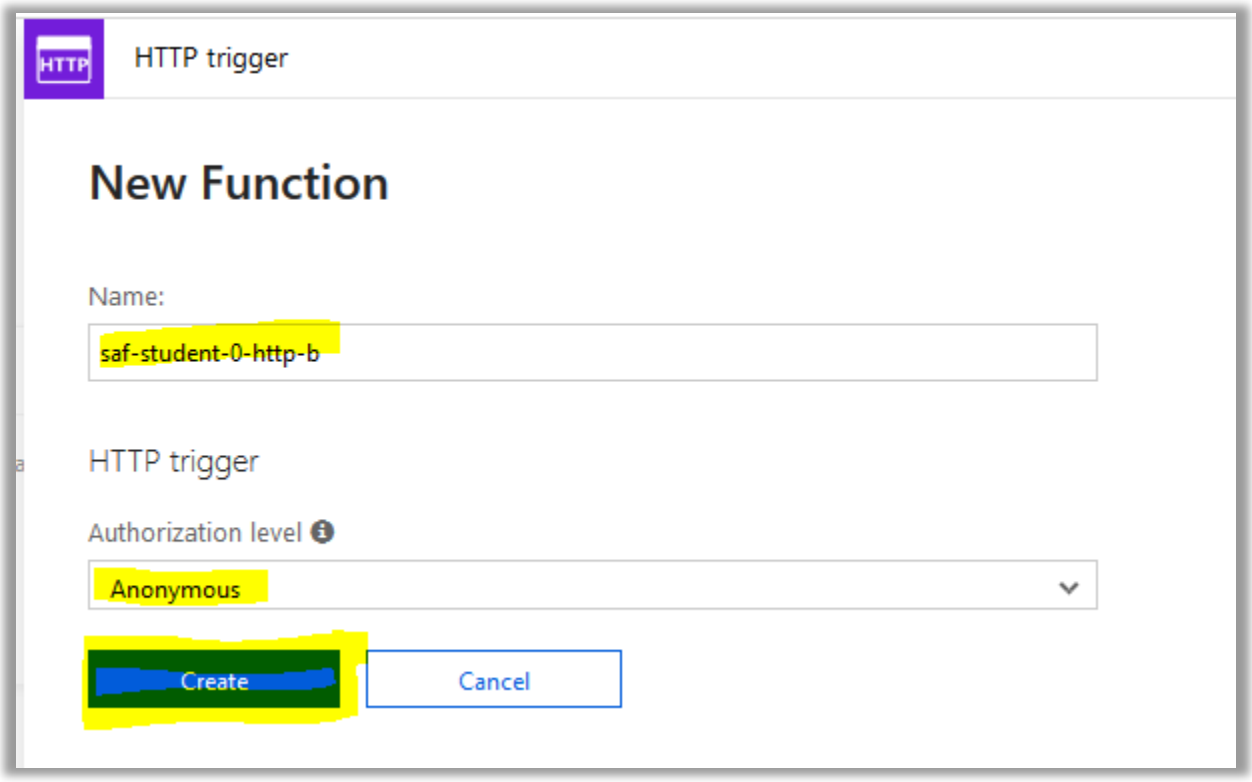
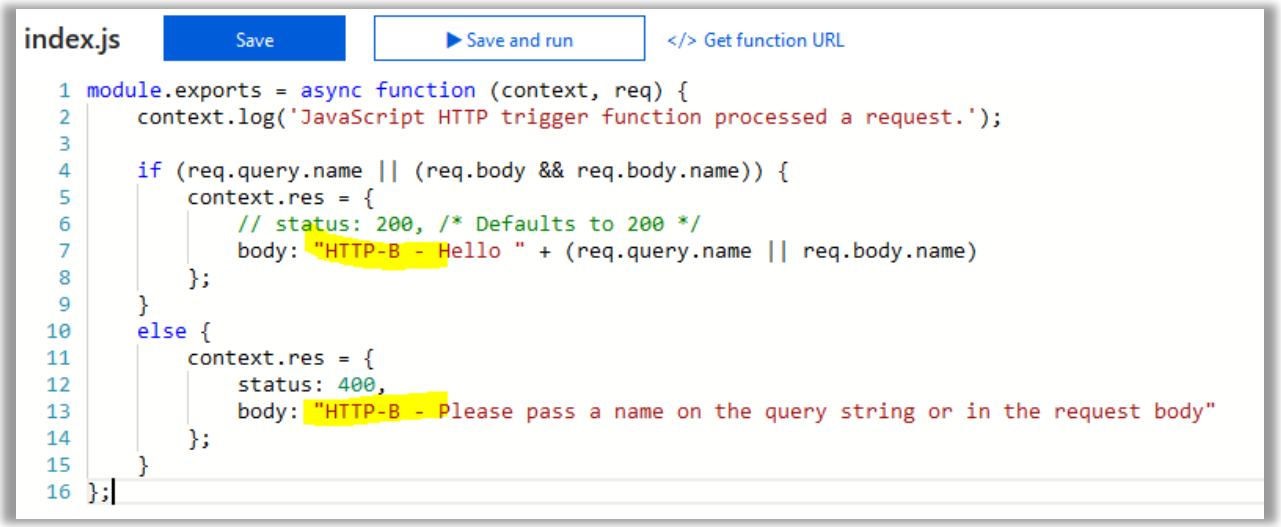
Lab 3: Use Azure API Management to create a Canary deployment with Azure Functions

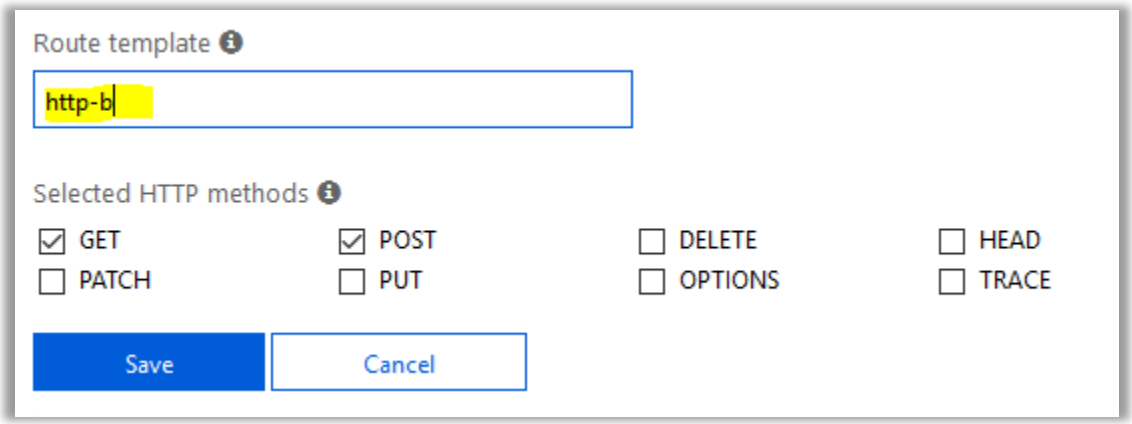
Create a new HTTP function following the same process we did for the first lab. Name this one B or #2 so that we can tell it apart from the first function we created. This time we will set the authorization to Anonymous when we create the function instead of changing it after:



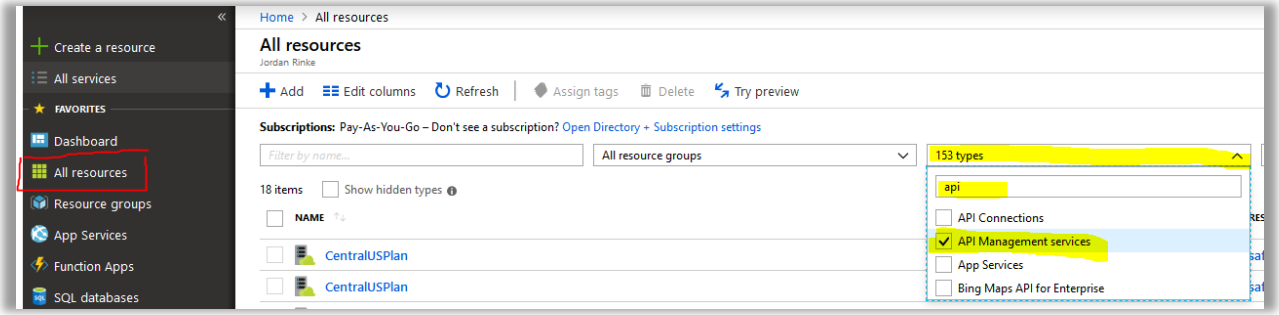
Edit the code of your new function to display a different message so we can tell the two functions apart. Leave the requests and logic the same, otherwise our function might randomly break as one or the other is selected:

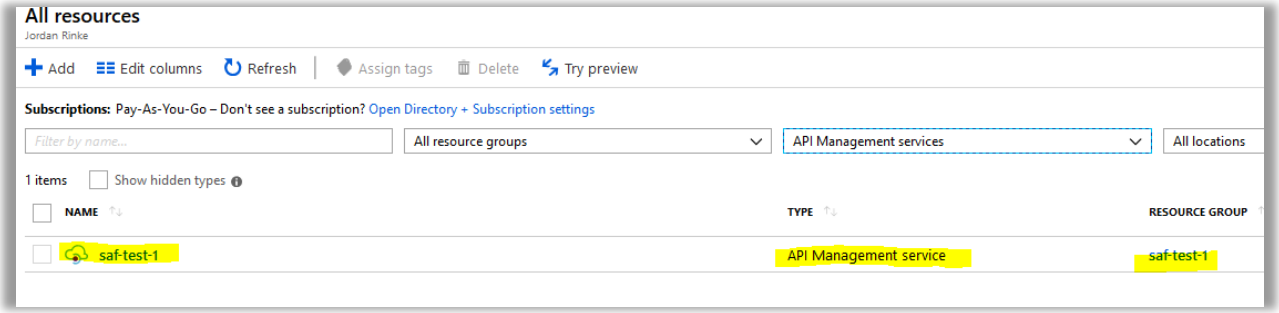


Give this function a route template that identifies it as the counter part to our function from the first lab:

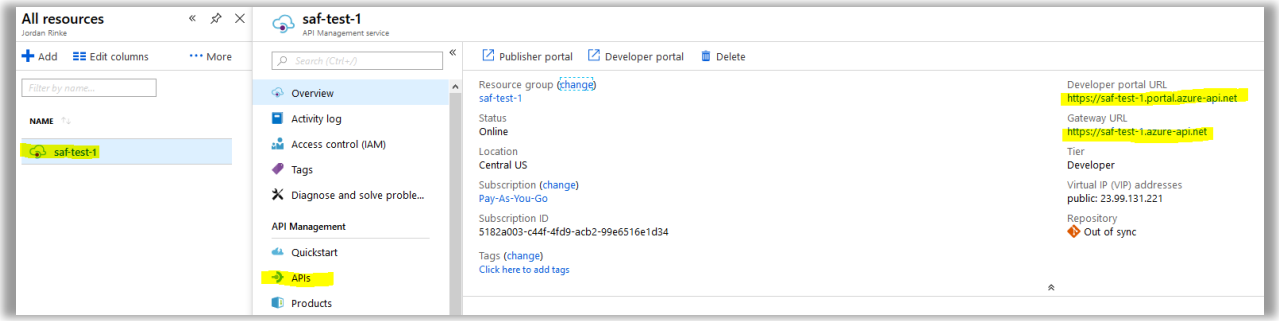


The API Management service is not listed on the left so we need to find it. Click on all resources and change the type to be API Management Services. You can use the search to find it since the list is so long. Your instructor will provide you with the name of the API Management service to be used for this course, it will likely have a different name than the sample below:

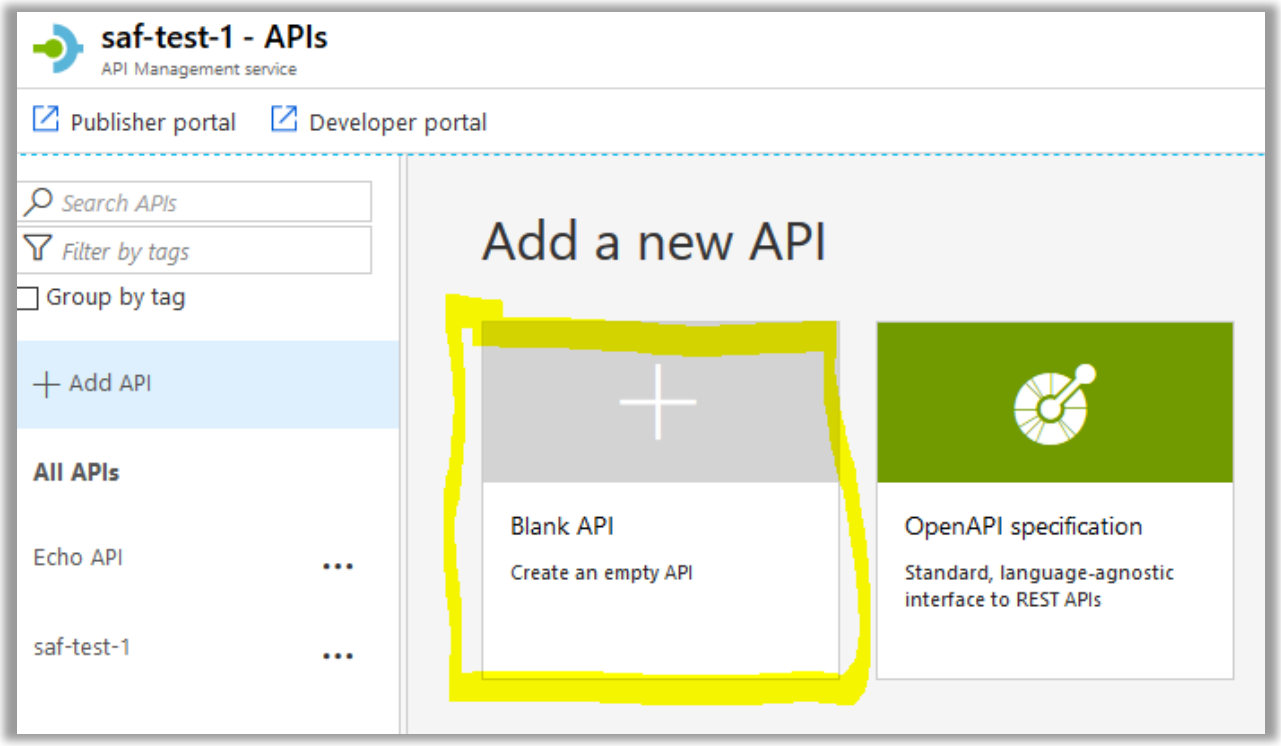




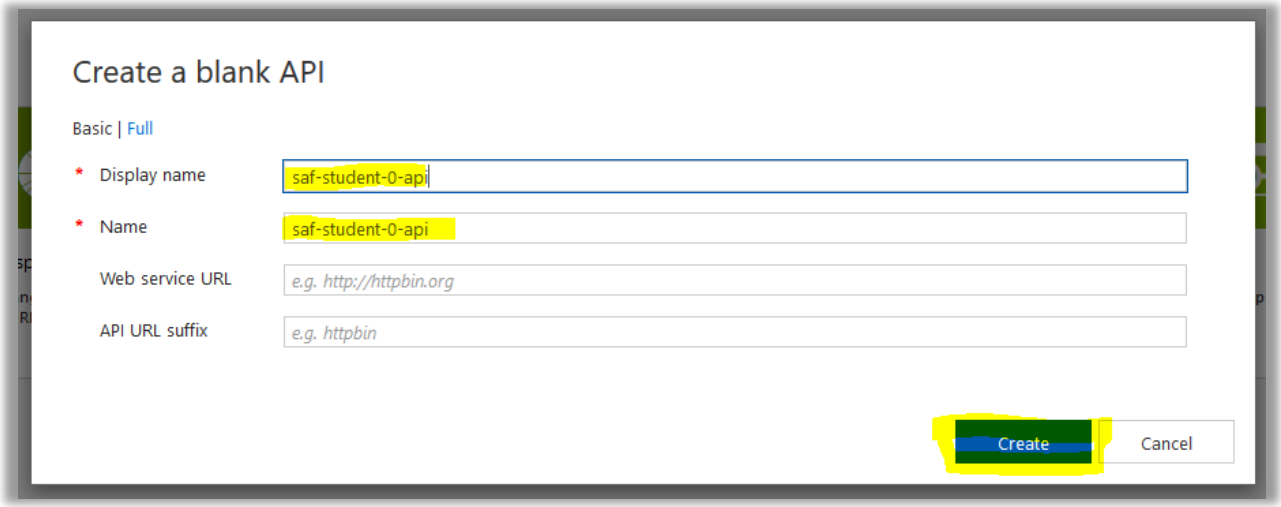
Click on the API Management Service. Make a note of the Gateway URL. We will need to use this to test our API once we create it. Click on "APIs":



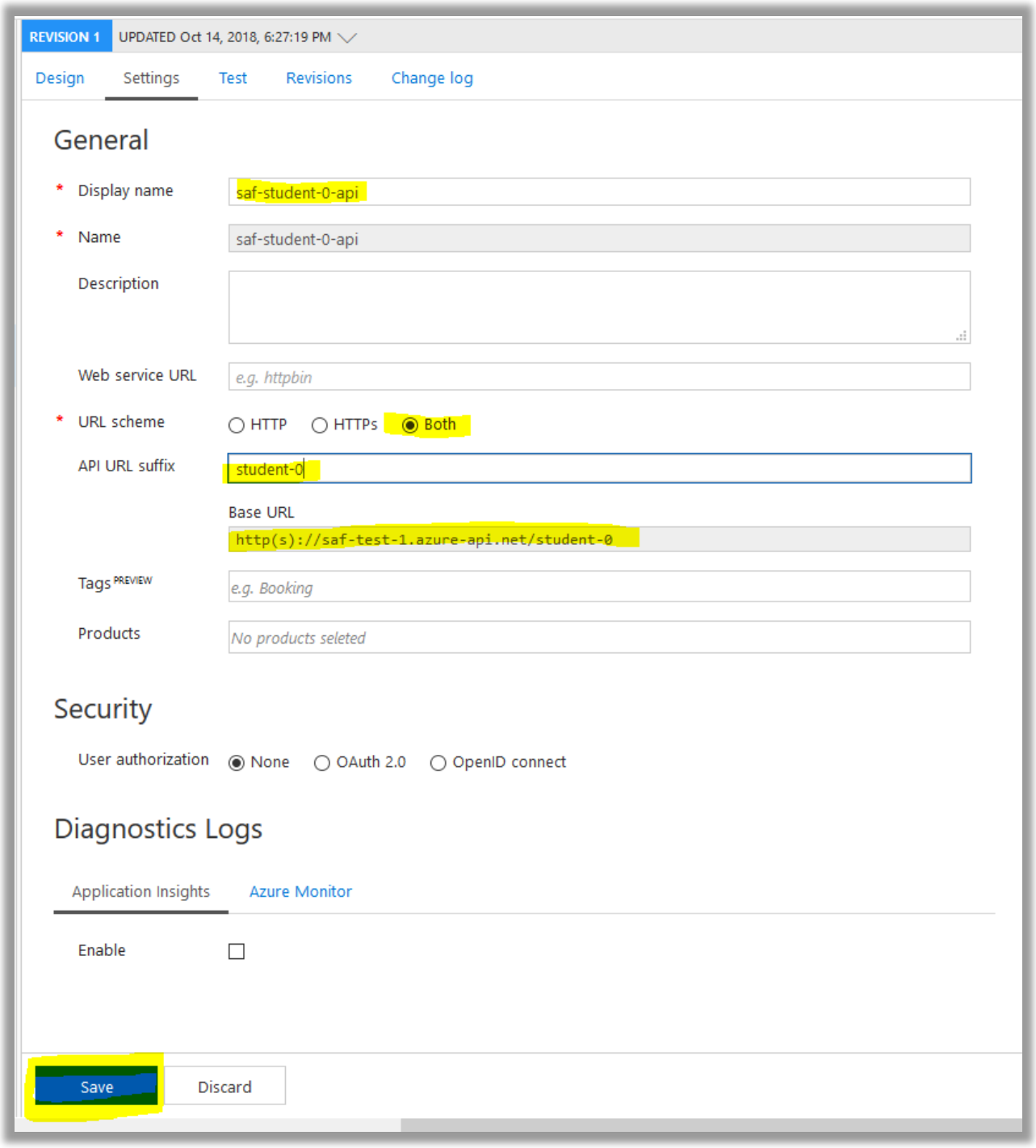
Multiple APIs may be listed already, click on “Blank API” to create a new API:



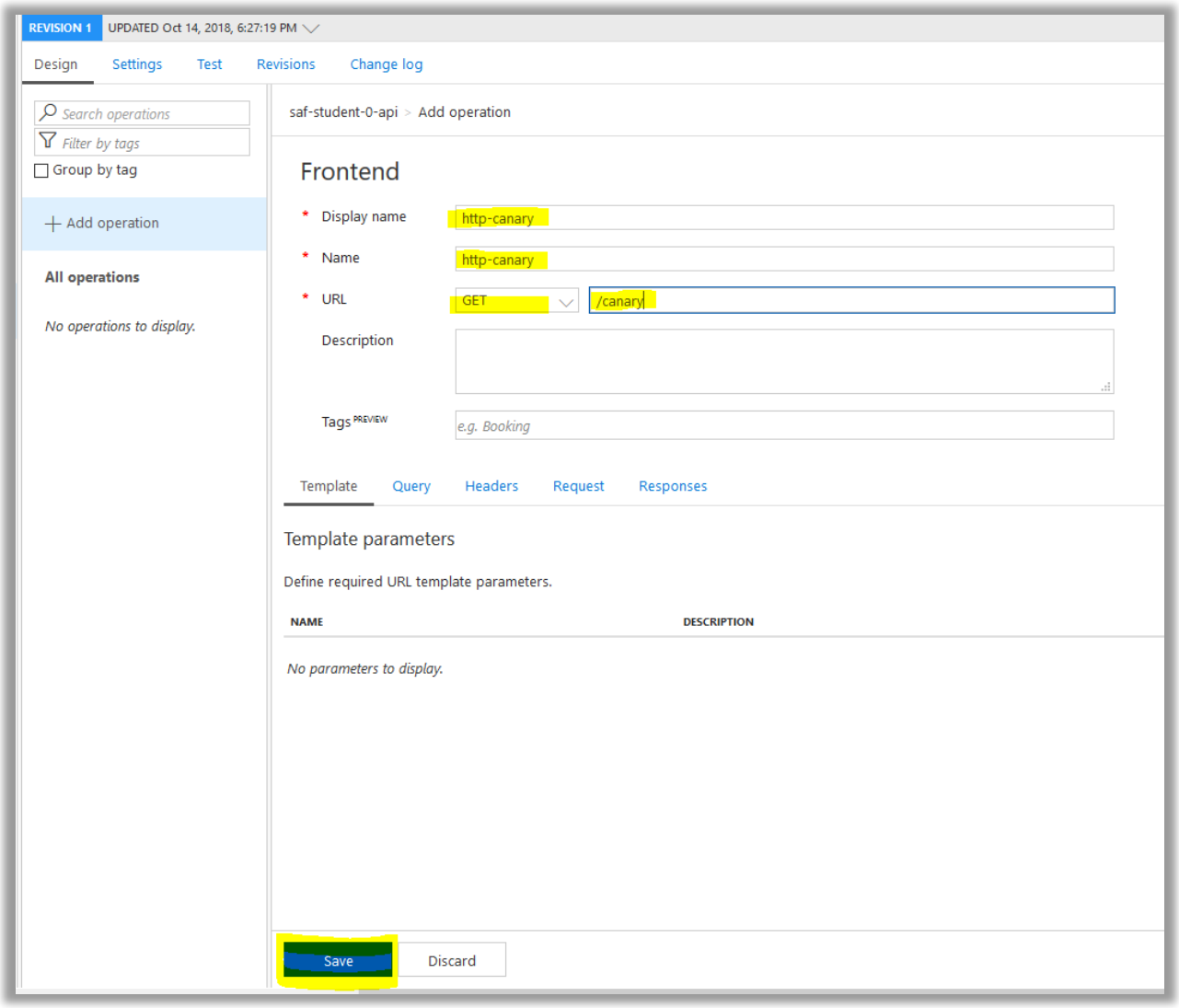
Give your API a unique name based on your student number:



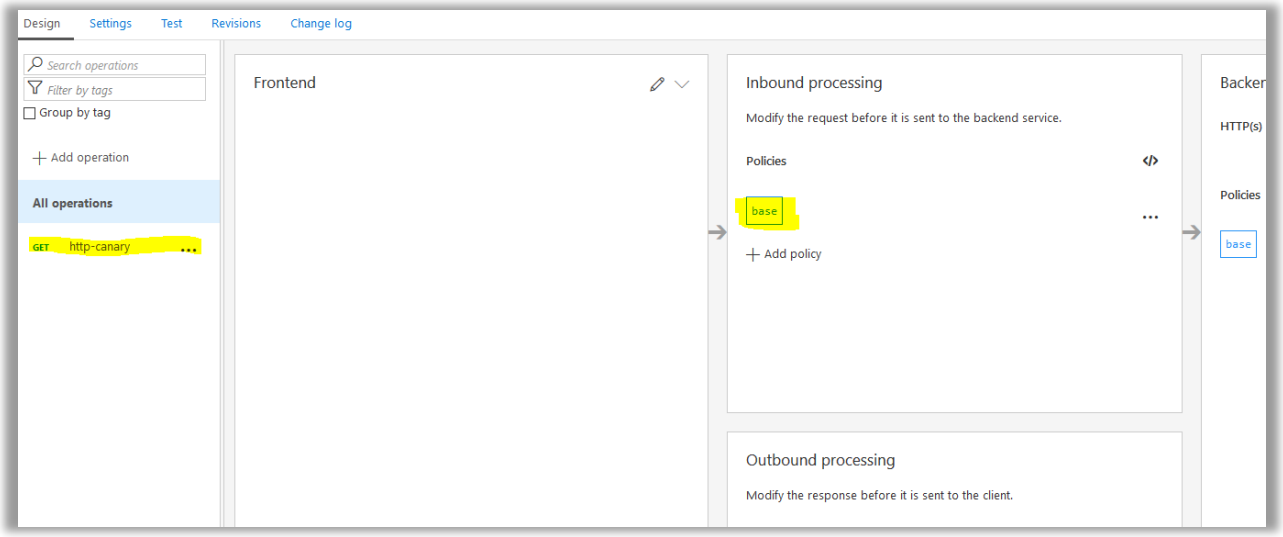
Go to the settings of your new API. Create a URL suffix, and change the Schema if you want to enable HTTP and HTTPS. The suffix is shown appended to the base URL below where you create it. Make a note of this. We will use this, plus the operation we are about to create to access our back end functions. Click on “Save”:



Go back to the “Design” tab of your new API. Select “Add operation”. Give your operation a meaningful name, it only needs to be unique to this specific API. Select GET for the URL and provide a path. In this case specifying “/canary” means that our URL base above, plus our operation URL will result in “https://saf-test-1.azure-api.net/student-0/canary”. Click on “Save”:

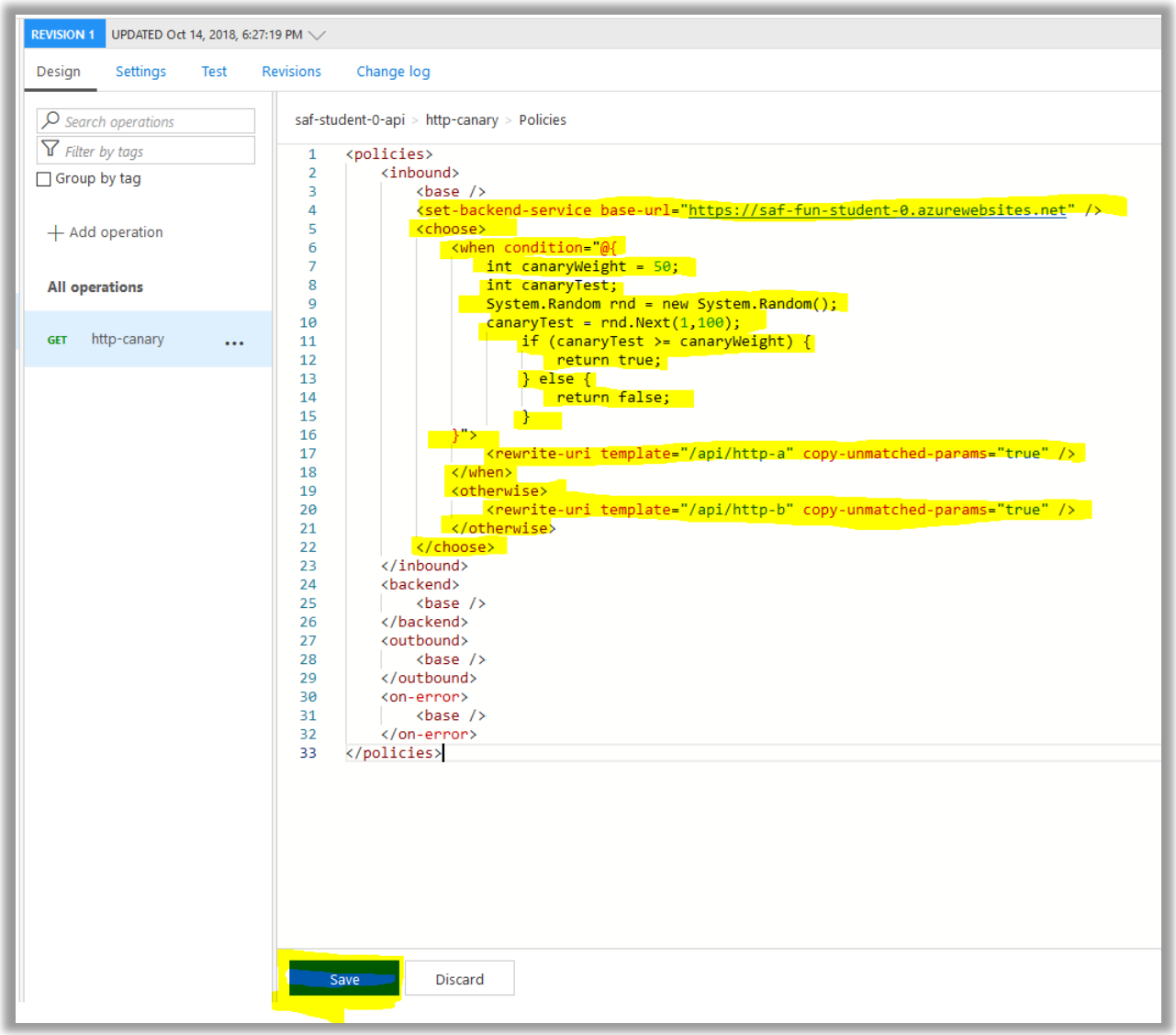


Click on the new operation that was just created. You will see a visual policy planner similar to the API Gateway from AWS. Clicking any of the boxes will take you to the advanced editor. You can visually edit all of the policies but since we are doing a very custom interaction we will use a code snippet to define all of our policy configuration. Click on “Base” on the inbound processing pane:

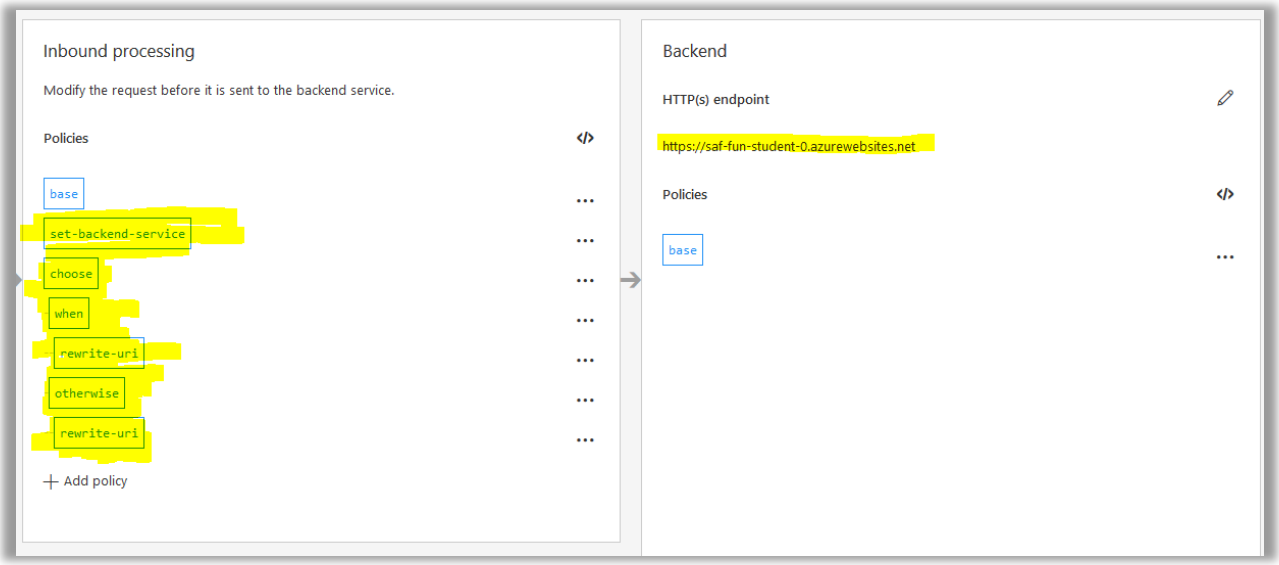


Azure doesn't support the simple Canary deployment options that AWS does but it is possible to extend routing decisions using C# code. We will first set the back end service to be the URL from the first lab, this is the root URL for the back end. We will then use C# code to check a random number between 1-100 to see if it is equal or larger than 50 (effectively creating a 50/50 canary). We use this in a choose statement, if the value is above 50 we re-write the URL to point to our first function, if it is below, we re-write the URL to point to our second function. This can be customized much more extensively and managed via a RESTful API – there are extensive documents describing how the routing logic works but it is does require a fair bit of learning curve. Use the following code to define our custom canary behavior and click "Save":

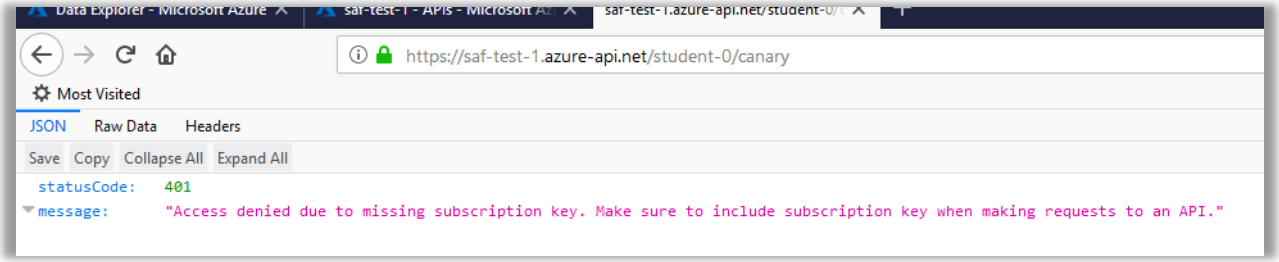
```
<set-backend-service base-url="https://saf-fun-student-0.azurewebsites.net" />
<choose>
  <when condition="@{
    int canaryWeight = 50;
    int canaryTest;
    System.Random rnd = new System.Random();
    canaryTest = rnd.Next(1,100);
      if (canaryTest >= canaryWeight) {
        return true;
      } else {
        return false;
      }
  }">
    <rewrite-uri template="/api/http-a" copy-unmatched-params="true" />
  </when>
  <otherwise>
    <rewrite-uri template="/api/http-b" copy-unmatched-params="true" />
  </otherwise>
</choose>
```


REVISION 1
UPDATED Oct 14, 2018, 6:27:19 PM
Design
Settings
Test
Revisions
Change log
Search operations
Filter by tags
Group by tag
Add operation
All operations
GET
http-canary
saf-student-0-api > http-canary > Policies
<policies>
<inbound>
<base />
<set-backend-service base-url="https://saf-fun-student-0.azurewebsites.net" />
<choose>
<when condition="@{
int canaryWeight = 50;
int canaryTest;
System.Random rnd = new System.Random();
canaryTest = rnd.Next(1,100);
if (canaryTest >= canaryWeight) {
return true;
} else {
return false;
}
}">
<rewrite-uri template="/api/http-a" copy-unmatched-params="true" />
</when>
<otherwise>
<rewrite-uri template="/api/http-b" copy-unmatched-params="true" />
</otherwise>
</choose>
</inbound>
<backend>
<base />
</backend>
<outbound>
<base />
</outbound>
<on-error>
<base />
</on-error>
</policies>
Save
Discard

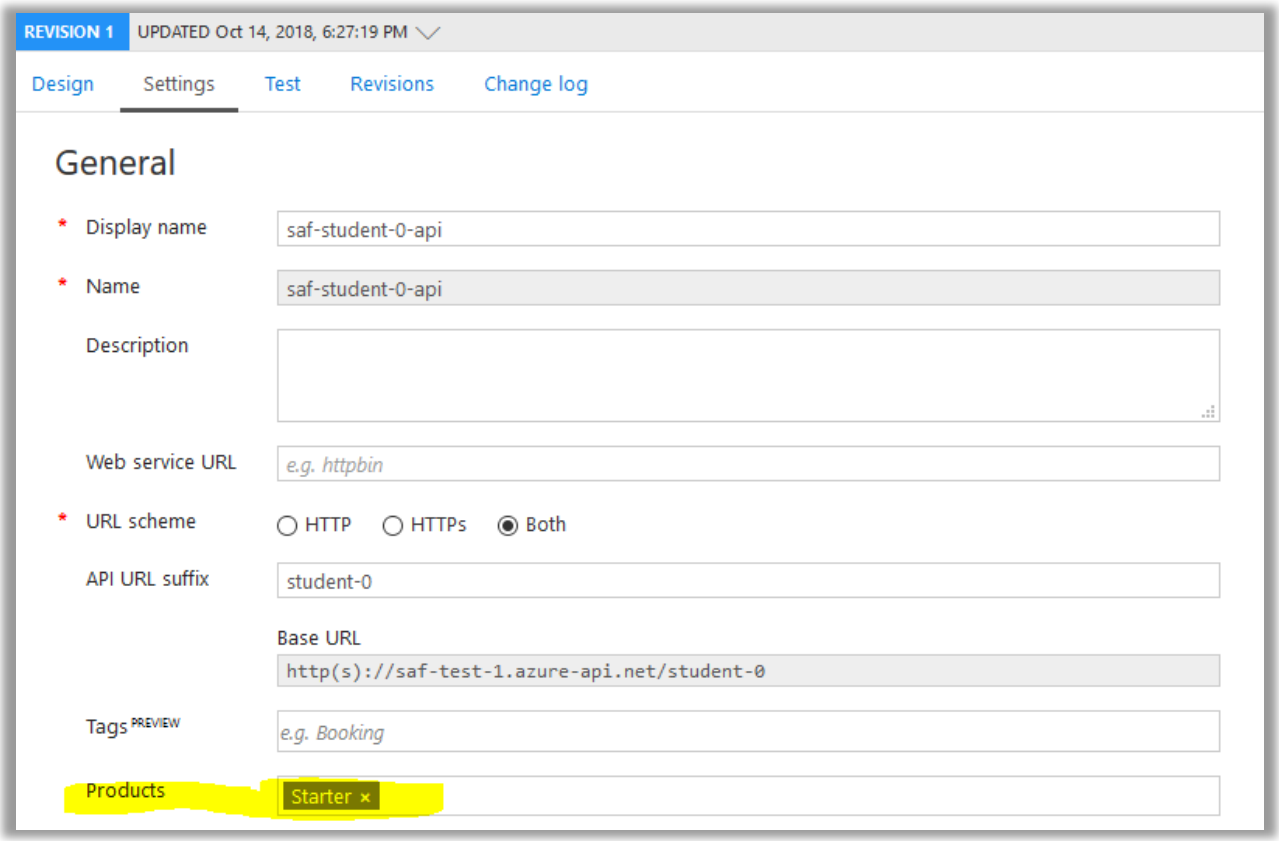
You will see the changes we made are now reflected on the visual policy designer:



If you attempt to access the new API we have created, you will get a 401 error. Our functions were set to anonymous, so why are we getting an access error?



The API Manager is designed for releasing full “products” tied to specific versions. You will need to add a defined product. Azure comes with two built in products, Starter and Unlimited. They define a few base rate limiting options. Apply the “Starter” product in the settings tab for the API that you created:



Now use the URL to access your API – after a few tries you should see it show you both versions of the function from the URL you created with the GET operation. The “Starter” product is rate limited to 5 requests per minute by default, you will see it if you hit the rate limit: