Onboarding Process

This document describes the steps for an engineer that is joining the Microservices team, step by step.

Get Access

You must immediately talk with your team lead, as well as with PSE’s Dan Marquett and/or Ekta Sachan to obtain:

1. A PSE account;
2. A VDI machine;
3. Access to the Microservices repository (which is where this wiki resides);
4. Access to the Microservices General group at Cisco Spark (see “Tools you will need”, below).
5. Access to the internal NuGet package source. (**Make sure you get assigned a package management license in TFS otherwise you will get 401 errors when restoring packages**)

Obtaining the PSE account is a morose process, it can take from 1 to 4 weeks.

You will not be able to do much of what’s listed here before you get your PSE account

So, get your PSE account.

Network access

Before you get a PSE account, you will use the PSE\_GUEST wifi network. This network uses a password that changes monthly, ask you team lead for this password.

After you obtain your PSE account, you should use the psewifi wifi network.

Tools you will need

* Cisco Spark: used extensively for communication among the team. Get it from here: <https://web.ciscospark.com/download>.  
  Web version here: <https://web.ciscospark.com/rooms/d406a510-e5eb-11e7-86dd-f1ec4b33acd6/chat>
* VMWare Horizon Client: used to connect to your VDI machine. Get it here: [https://my.vmware.com/web/vmware/info?slug=desktop\_end\_user\_computing/vmware\_horizon\_clients/3\_0](https://my.vmware.com/web/vmware/info?slug=desktop_end_user_computing%2Fvmware_horizon_clients%2F3_0)
* Visual Studio: [IDE Setup](http://awowttfap01v01:8080/tfs/WebAndMobile/Microservices/Microservices%20Team/_wiki?pagePath=%2FDeveloper-Quickstart%2FIDE-Setup)
* [Dotnet Core](https://github.com/dotnet/core)
* [Git](https://git-scm.com/)
* [NuGet](https://docs.microsoft.com/en-us/nuget/install-nuget-client-tools)

You will also need to be acquainted with [Swagger](https://swagger.io/), the tool for REST API definition and documentation.

Use the VDI machine

From now on, all instructions should be executed in your VDI machine.

TFS Repository

This is the TFS repository for the project:

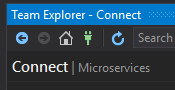
<http://awowttfap01v01:8080/tfs/WebAndMobile/Microservices>

Follow these steps to

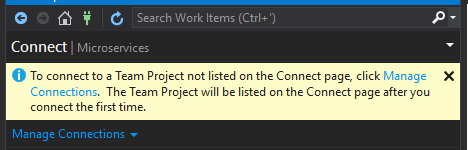
1. Add the internal nuget package source. Open a command prompt and execute:

nuget sources add -Name “PSE MicroServices” -Source <http://awowttfap01v01:8080/tfs/WebAndMobile/_packaging/testfeed/nuget/v3/index.json>

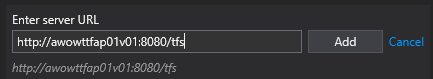
1. After you confirm you have access to that server, you can connect to the TFS project in Visual Studio: Open Visual Studio (it must be Professional or Enterprise in order to have access to the TFS project);
2. Open the Team Explorer: go to View / Team Explorer;
3. Click on Manage Connections, the green “power plug” button



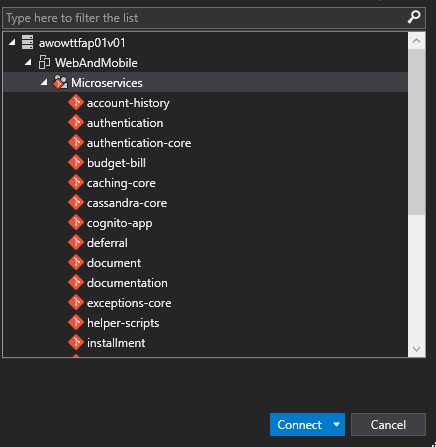
1. It will show a message, click on “*Manage Connections*” link



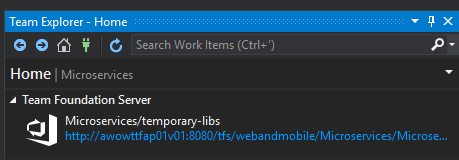
1. Click on *Add TFS Server*
2. Enter <http://awowttfap01v01:8080/tfs>, then click *Add*

**

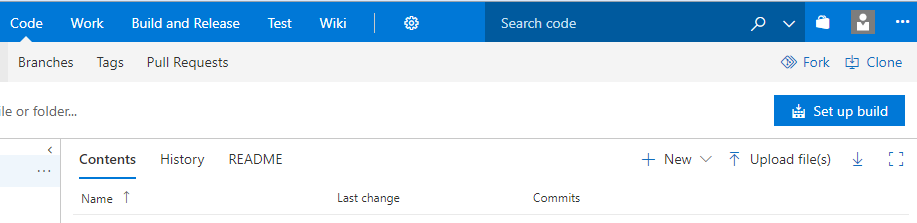
1. Select “*Microservices*”, then click on Connect



1. Now, let’s clone the code. At the Team Explorer pane, click at the Home icon, then at the link under *Team Foundation Server*:



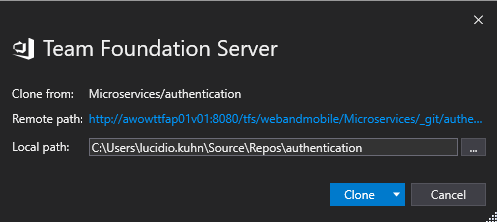
1. You will be directed to the TFS page at your browser:



1. Click at the “*Clone*” button at the top-right, then select *Clone in Visual Studio*:



1. Eventually, Visual Studio will open, asking you where you want to clone the repository to. Enter your preferred code location, or just use the default, then click *Clone*.



Congratulations, you have your source code. Now, get back to work!

Swagger definitions

Swagger definitions are located in the [documentation](http://awowttfap01v01:8080/tfs/WebAndMobile/Microservices/_git/documentation) repository. Under folder *src/wwwroot/docs*, you will find all Swagger definitions for all stories.

Branching policy

1. The project uses git for source control and branching.
2. Main branch name is *master* .
3. All commits must be executed against a branch that was created for a specific TFS Task.
4. A such, no commits should be performed on the *master* branch.
5. Branches for Tasks must be names *feature/task\_name*, forked directly from the *master* branch.
6. Commit your changes, then push them to the *feature/task\_name* branch\_ (See *Development workflow*, below).

Please follow the instructions on [Branching Strategy](http://awowttfap01v01:8080/tfs/WebAndMobile/Microservices/_wiki?pagePath=%2FMicroservices%2FDevelopment-Process%2FBranching-Strategy).

Development workflow

The basic workflow goes like this:

Pick a Story from the Backlog;

Create a Branch associated with the Story, forking directly from master;

Create Tasks associated with the Story (at least Investigation, Unit Tests, Development, Documentation);

For each Task:

Implement the Task, performing commits and push as you go;

Pull and merge from master;

Run Unit Tests, make sure they pass;

Create Pull Request, assigning code reviewers for your changes, use Auto Complete;

Implement changes suggested during the code review;

When Pull Request is approved by three reviewers, it will be merged automatically to master.

An example

If you want to follow an example, take a look at Story [51 Payment Arrangement: Active Arrangement Details](http://awowttfap01v01:8080/tfs/WebAndMobile/Microservices/_workitems?id=51&_a=edit).

This story has four tasks associated with it:

* Investigation;
* Unit Tests;
* Development;
* Documentation.

If you, like me, prefer to look at code, look at this pull request:

[http://awowttfap01v01:8080/tfs/WebAndMobile/Microservices/Microservices Team/\_git/installment/pullrequest/307](http://awowttfap01v01:8080/tfs/WebAndMobile/Microservices/Microservices%20Team/_git/installment/pullrequest/307)

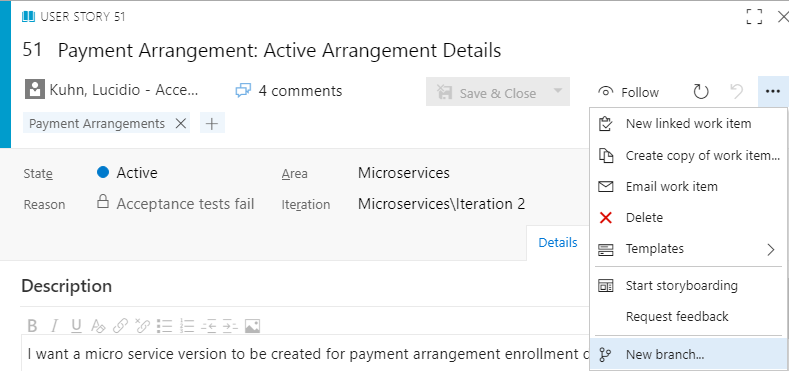
Pick a Story from the Backlog

Here’s the backlog:

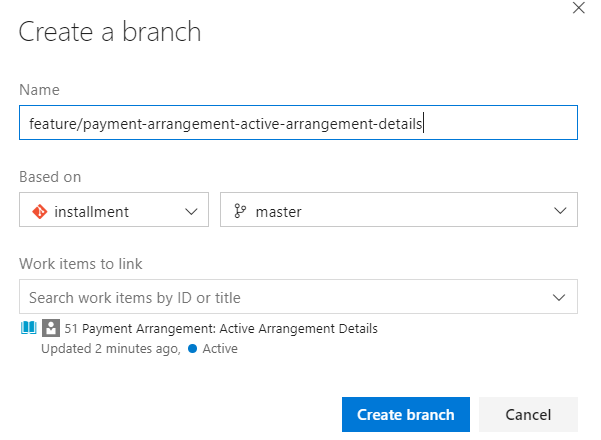
<http://awowttfap01v01:8080/tfs/WebAndMobile/Microservices/_backlogs?level=Stories&showParents=false&_a=backlog>

Create a Branch associated with the Story

In the Story dialog, select “*New Branch*”, under the “…” menu:

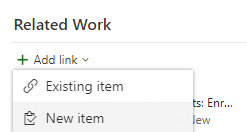


The branch name should be *feature/the-story-name*, such as:

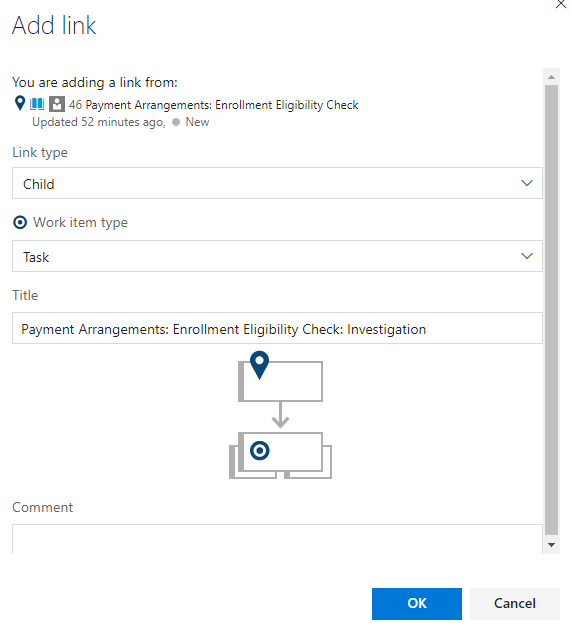


Create Tasks associated with the Story

Open the Story, then, under *Related Work*, *select Add Link* / *New Item*.



On the Add Link page, select Work Item Type Task, and enter a decent name, such as Name\_Of\_Story: Name\_Of\_Task



Again, it is advisable to create at least 4 tasks: one for Investigation, one for Unit Tests, another Development and one for Documentation.

Implement the Task, performing commits and push as you go

Implement the task, pulling and merging from master (see next session). It is a good idea to also commit frequently and push the changes frequently, to avoid mishaps.

Pull and merge from master

Do not forget to frequently:

* Pull and merge from the parent branch. For instance, if you are using branch *feature/payment-arrangement-active-arrangement-details* and the parent branch is *master*:  
  git pull origin master
* You can solve conflicts using SourceTree, or if you want to do it manually:

run git status --porcelain`

for every line that starts with 'UU', 'DU', 'UD', 'AA', 'AU' or 'UA':

edit the file,

search for <<<< (this is the marker for conflict)

for each conflict:

there will be 2 texts in conflict:

what is between the line <<< and the line === is your text,

between === and >>> is the text that was brought from the parent branch.

you can edit the conflicted text, do not forget to remove the <<<, === and >>> lines.

save the edited file,

got back to the command prompt,

run git checkout the/file/you/have/just/edited

Run Unit Tests, make sure they pass

Before creating a pull request, make sure

* Your code is merged with master,
* Your code builds,
* Your code does what is supposed to do: at least once run the code with the debugger,
* All Unit Tests pass,

Create Pull Request, assigning code reviewers for your changes, use Auto Complete

Implement changes suggested during the code review

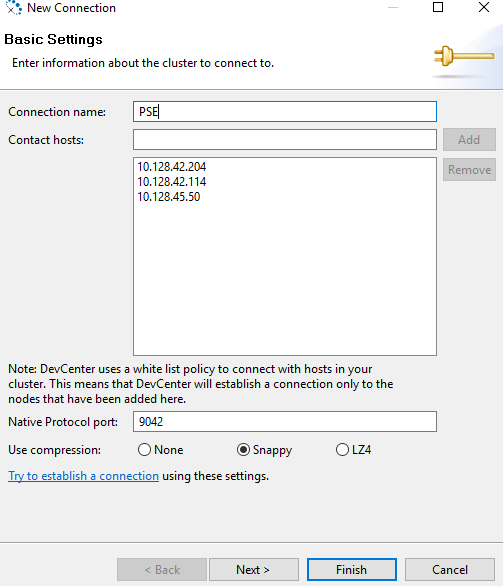
Suggestions from code reviews must be implemented, unless you have strong evidence not to.

When Pull Request is approved by three reviewers, it will be merged automatically to master

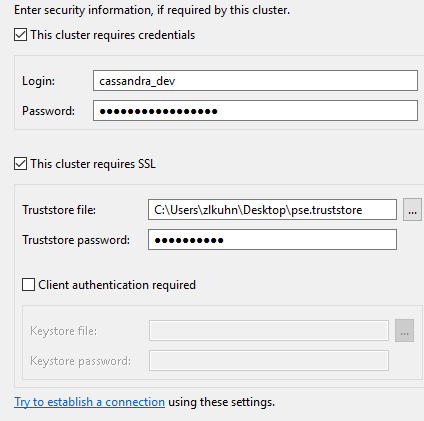
**Accessing Cassandra DB**

At the VDI machine, follow these steps:

1. Open File Explorer, go to \_\corpnas02b\pse\ekta\_
2. Copy the file *pse.truststore* from \_\corpnas02b\pse\ekta\_ to your desktop.
3. Open directory *“\corpnas02b\pse\ekta\Cassandra Dev Setup”*
4. Double click on file *DevCenter-1.6.0-win-x86.zip*, to enter in this zip file.
5. Copy folder *DevCenter* from the zip file into your desktop.
6. Open another File Explorer, go to *“C:\Program Files (x86)\Java\jre1.8.0\_152\lib\security\policy”*
7. Just to be safe, copy and paste both folders (*limited* and *unlimited*) in place (Control-A, Control-C, Control-V).
8. Go to directory *“\corpnas02b\pse\ekta\Cassandra Dev Setup\jce\_policy-8\UnlimitedJCEPolicyJDK8”*
9. Copy files *local\_policy.jar* and *US\_export\_policy.jar* from *“\corpnas02b\pse\ekta\Cassandra Dev Setup\jce\_policy-8\UnlimitedJCEPolicyJDK8”* to *“C:\Program Files (x86)\Java\jre1.8.0\_152\lib\security\policy\limited”* and to *“C:\Program Files (x86)\Java\jre1.8.0\_152\lib\security\policy\unlimited”*.
10. From a Micro Services git repository, open file *src\localConfiguration.json* with your preferred editor.
11. On File Explorer, go to *Desktop* / *DevCenter* and double-click *DevCenter.exe*
12. Select *New* / *Connection*.
13. Give it a decent name, then copy the IP Addresses of each Cassandra Host from *src\localConfiguration.json* into “Contact Hosts” and click “*Add*”. You will end up with three contact hosts on the list.



1. Click on *Next*
2. Enable both “*This cluster requires credentials*” and “*This cluster requires SSL*” and fill in the fields:  
   *Login* with value of “*ClusterUser*” from *src\localConfiguration.json*  
   *Password* with value of "*ClusterPassword*"  
   *Truststore file:* select the *pse.truststore* file that was saved on your desktop.  
   *Truststore password:* *HotSoup123*



1. Click on *“Try to establish a connection”*, hopefully everything is well.

**Branching strategy**

The move to Git and Microservices project brings opportunities for simplification in many areas. This wiki lays out a framework for branching based on best practices gleaned from years of working with Git.

OneFlow is a Git branching model that addresses many common challenges. It reduces the number of branches and extends the idea of the single eternal (“master”) branch that many of us are accustomed to. This works well for microservice development partly because each microservice is a separate repository which will have a very limited number of developers. The simplified branching scheme described in the OneFlow method reduces the complexity of merge actions by keeping the scope of each branch small and tied to a specific purpose.

The Oneflow blog explicitly lays out the commands that should be used for common tasks. This provides a common language and approach that should be used by all involved teams.

It is described in detail here:  
[OneFlow blog](http://endoflineblog.com/oneflow-a-git-branching-model-and-workflow)

Each of the branches below is described on the oneflow blog. Please refer to the blog for the specific commands to use

The development teams will arrange the branches in their repositories as follows:

* “master” - The main branch. Feature branches are branched from here and merged back here after review and approval
* “feature/<feature-name>” - The branches where where the day-to-day development work happens. These should be created as-needed and destroyed after merging back into master. The merge into master should follow the procedures detailed in “Finishing a feature branch”, option #3 of the [OneFlow blog](http://endoflineblog.com/oneflow-a-git-branching-model-and-workflow).
* “release/<version-number>” - Release branches are created to prepare the software for being released. The version number will follow [semantic versioning](https://semver.org/#summary)
* "hotfix/<version-number> " - hotfixes will be tied to a specific release version and will increment the patch number

**Creating Cognito Users for MCF**

You will have to work with Vikas to setup the data in MCF. One thing to note is you will have to make sure the BP’s associated with the accounts you listed have an email as well as BP –Email combination we have in cassandra should be similar to what they have in MCF.

Before you run the below make sure .net core is running in your machine  
Steps

1. Get MCf Data from Vikas with the format AccountNumber,BP, Email
2. Run query number 1 to check if the contact account and bp exists and
3. then run query #2 to see if the Bp and email match up(replace the actual bp from MCF data).
4. If they don’t match ,run query # 3 to insert data to make sure BP and Email match.
5. Copy the the file under I:\Webteam\DropBox\yared\CognitoUserCreation
6. Run the exe under the shared location passing username,bp, email like in Query #5. If it is successful it will give you the cognito id, if it fails it will give you the error message
7. Run the query # 4 in the Cassandra self\_service key space in the user\_auth table replacing the highlighted with the username, bp and email above

Query #1,2,3 should run in the microservices keyspace  
Query # 4 should run in the selfservice\_auth

Query #1 (Replace the contaact coming from MCF team)  
select \* from contract\_account where contract\_account\_id IN (20000044574, 20000044575) allow filtering;

Query #2(replace with the Bp coming from MCF team)  
select \* from customer\_contact where bp\_id IN(1201108586,1001622788,1004063101,1201195610,1201195578);

Query #3(replace BP and email from MCF Data coming)  
INSERT INTO customer\_contact (bp\_id,email,mailing\_address,phones) VALUES (1201195578,‘cassandra@gmail.com’,null,{‘cell’:{number:‘5097686990’,extension:NULL}});

Query #4  
INSERT INTO user\_auth (user\_id,bp\_id,cognito\_id,email,failure\_count,last\_attempt,last\_login,legacy\_salt,lock\_time,locked,password\_hash,password\_type,user\_name) VALUES  
(uuid(),1003710092,7901a31e-1f11-438a-ba4f-70a9d767fbf6,‘test7@test.com’,0,1517264993396,1517264993396,‘thee1222’,1517264993396,false,’$argon2id$v=19$m=32768,t=4,p=1$ROiGp4NKJARgZZwwyVnnzQ$2Mja37it6sXcras7Y7Nc9Brwi1GwQXnrbdMo5D/Y0SQ’,0,‘testuser7’);

Query #5  
C:\Users\zygebre\source\repos\poc\bin\Release\netcoreapp2.0>dotnet cognitopoc.dll “testusercreation3” “12345699877” ["testemail@test.com](mailto:%22testemail@test.com)"  
Hello World!  
the cognito Id is e8868e4d-63d8-4f61-8d74-52e59601d30d

Cont.Account Bus. partner Email

220013890656 1201108586 JENNIFER@gmail.com

220015113446 1001622788 connie@gmail.com

200028594063 1004063101 josephs@gmail.com

220015135985 1201195610 amyw@gmail.com

220015136157 1201195578 cassandra@gmail.com