Tray.io Technical

Why the Program times out:

This could be a result of the application still pointing/reading from the legacy health check flags or deploying the applicat ion without enabling the updated health checks first.

The front end (Web Tier) sending messages into a queue for the back end (App Tier) to process has issues with resource conten tion such as High CPU, RAM, Swap Space utilisation.

The endpoints could be designed to return HTTP status code 202 accepted if a requested resource is not found in cache
HTTP status code 200 OK is returned if requested resource is found in cache

We can also check that Firewall access is in place for the new environment.

To Fix:

Make sure the legacy health_check flag does not exist in the app.yaml file

Run …. App describe to check if splitHealthChecks flag is set to true under featureSettings

Ensure endpoints are configured to accept multiple return types so you can overload response types

Create a firewall rule on ASA firewall and/or Barracuda.

To monitor/analyse resource contention I would use NetData to review metrics such as:

CPU - steal, softriq, user, system, iowait

Load - 1, 5 & 15 minute averages

Disk - out k/b

Ram - free, used, cached & buffers

Network - received, sent

Processes - System processes. Running are the processes in the CPU. Blocked are processes that are willing to enter the CPU, but they cannot, e.g. because they wait for disk activity.

Context Switches - too many processes willing to execute against very few CPU cores available to process causes more content switching which is CPU intensive and the slower the system becomes.

We can also look to increase CPU, RAM and Swap Space (where the Swap space allocated is dependent on the amount of installed RAM).

We can also use Decoupling to utilise Dual queue messaging:

A queue for requests coming in
A queue for requests going back out to the client:
    Use AWS SQS for Azure Queues for Cloud Environment

Use RabbitMQ for DataCenter hosted Infrastructure

To analyse the application from the url to the line of code I would use ManageEngine Applications Manager

(Building a Production Infrastructure for the application)

Using Azure and Flynn

1. Set up a Flynn Cluster in Azure

Run Flynn install script:

```
sudo bash < <(curl -fsSL https://dl.flynn.io/install-flynn)
```

The installer script will install Flynn runtime dependencies

Download verify and install the flynn-host binary
Download and verify filesystem images for each Flynn component(s)
Installs an Upstart job for controlling the flynn-host daemon

Complete the above steps on every host that you want to include in the Flynn cluster

(Set up Nodes)

ensure all network traffic is allowed between all nodes in the cluster (specifically all UDP and TCP packets).

The following ports should also be open externally on the firewalls for all nodes in the cluster:

- 80 (HTTP)
- 443 (HTTPS)
- 3000 to 3500 (user-defined TCP services, optional)

Configure a Layer 0 cluster by starting the flynn-host daemon on all nodes.

As the daemon uses Raft for leader selection it needs to be aware of all of the additional nodes for it to function correctly .

n.b
If starting more than one node, the cluster should be configured using a discovery token. flynn-host init is a tool that handles generating and configuring the token.

On the first node, create a new token with the --init-discovery flag.

The minimum multi-node cluster size is three, and this command does not need to be run if you are only starting a single node.

```
sudo flynn-host init --init-discovery
https://discovery.flynn.io/clusters/53e8402e-030f-4861-95ba-d5b5a91b5902
```

On the rest of the nodes, configure the generated discovery token:

```
sudo flynn-host init --discovery https://discovery.flynn.io/clusters/53e8402e-030f-4861-95ba-d5b5a91b5902
```

(Start Flynn)

Start Flynn and confirm it has started

Ubuntu 16.04

```
sudo systemctl start flynn-host
sudo systemctl status flynn-host
```

Bootstrap Flynn

After you have running flynn-host instances, you can now bootstrap the cluster with flynn-host bootstrap.

You'll need a domain name with DNS A records
pointing to every node IP address and a second, wildcard domain CNAME to the cluster domain.

Example

```
demo.localflynn.com.   A     192.168.84.42
demo.localflynn.com.   A     192.168.84.43
demo.localflynn.com.   A     192.168.84.44
*.demo.localflynn.com.  CNAME  demo.localflynn.com.
```

Set **CLUSTER_DOMAIN** to the main domain name and start the bootstrap process, specifying the number of hosts that are expected to be present and the discovery token if you created one.

```
sudo \
   CLUSTER_DOMAIN=demo.localflynn.com \
   flynn-host bootstrap \
   --min-hosts 3 \
   --discovery https://discovery.flynn.io/clusters/53e8402e-030f-4861-95ba-d5b5a91b5902
```

n.b: You only need to run this on a single node in the cluster. It will schedule jobs on nodes across the cluster as required.

The bootstrapper will get all of necessary services running within Flynn. The final log line will contain configuration that may be used with the command-line interface.

2. Build the F# Suave Web App

Make sure you have Mono and **Visual Studio Code** with Ionide plugin installed.

We will use Ionide to create our project. Go ahead and create a directory e.g.
~/Projects/SuaveBootstrapFlynn and open it in VSCode. Open the Command Palette using **⌘⇧P** and search for **F#: New Project** and create the project with the name **SuaveBootstrapFlynn**.

Let's add Suave dependency. Open the **paket.dependencies** and make sure it looks like as follow

```
source https://www.nuget.org/api/v2
nuget FAKE
nuget FSharp.Core
nuget Suave
```

Let's also add Suave to **SuaveBootstrapFlynn/paket.references** which will take care of adding the dependencies references in your **.fsproj**.

Let's build a simple web app using Suave that outputs some HTML.

Edit **SuaveBootsrapFlynn/SuaveBootstrapFlynn.fs** as follows

```
module SuaveBootstrapFlynn
open Suave
open Suave.Successful
open Suave.Web
open Suave.Operators
open Suave.Filters
open System
open System.Net
open System.Threading.Tasks
let helloWorld _ =
   printfn "Saying hello world from F# and flynn %O" DateTime.UtcNow
   OK (sprintf "<html><body><h1>Hello World - %O</h1></body></html>" DateTime.UtcNow)
let app =
      GET >=> path "/" >=> request helloWorld
let config =
   let port = System.Environment.GetEnvironmentVariable("PORT")
   let ip127  = IPAddress.Parse("127.0.0.1")
   let ipZero = IPAddress.Parse("0.0.0.0")
{ defaultConfig with
      logger = Logging.Loggers.saneDefaultsFor Logging.LogLevel.Verbose
      bindings=[ (if port = null then HttpBinding.mk HTTP ip127 (uint16 8080)
               else HttpBinding.mk HTTP ipZero (uint16 port)) ] }

[<EntryPoint>]
let main argv =
   startWebServer config app
   0 // return an integer exit code
```

From the above code we can just get the PORT from the environment variables and start the server at that port else we start w ith port 8080.

3. Setting up for Deployment

Make sure you have initialized the project with Git.

In terminal run the following command to create a Flynn app, which also adds remote git repository for deployment.

```
flynn create suavebootstrapflynn
```

Add a **.buildpacks** file in the root folder of project, it includes the repository from which to pull the Heroku build pack

```
https://github.com/SuaveIO/mono-script-buildpack.git
```

Then create a file named Procfile and add the following to it:

```
web: mono build/SuaveBootstrapFlynn.exe
```

The **Procfile** is used to declare the commands that will be run by our application.

Here **mono build/SuaveBootstrapFlynn.exe** will run our application once it gets deployed.

4. Deploy

The next step is to deploy the app.

To do so, commit your changes and then push to flynn master

```
git push flynn master
```

Wait for the application to deploy and then run:
flynn info which should show the URL to your app. Go visit the url to verify the web app is running in Azure using Flynn.

Flynn can be used to scale your app, while taking care of deploying the servers and load balancing the requests across them.