Bootstrapper and Life Cycle Manager:

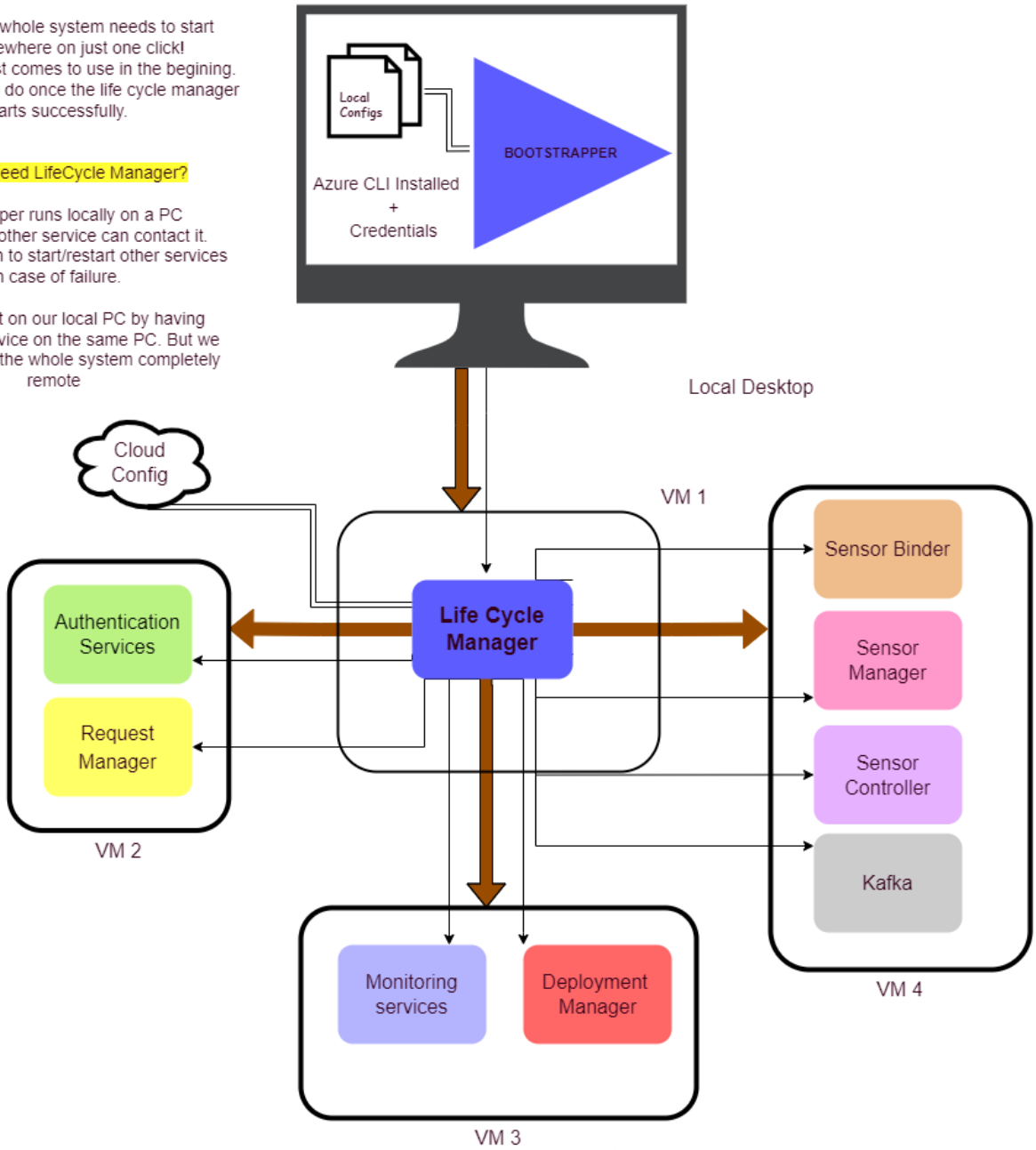
Why we need Bootstrapper?

Because the whole system needs to start from somewhere on just one click! Bootstrapper just comes to use in the begining. It has no work to do once the life cycle manager starts successfully.

Why we need LifeCycle Manager?

Bootstrapper runs locally on a PC therefore no other service can contact it. We need a twin to start/restart other services in case of failure.

We can do it on our local PC by having monitoring service on the same PC. But we want to make the whole system completely remote



- Similarities:
    - Both can provision a new VM(s).
    - Both Initializes VM environment for deploying docker containers.
    - Both Require Azure CLI as base.
    - Both require account credentials.

- Bootstrapper codebase is completely local, including config files.
    - Bootstarpper contains configs :
        - Required to login to Azure (details like, userID or subscriptionID)
        - (Optional) Provision a VM. (details like, VM type and configuration, etc.)
        - Details for accessing VMs. (details account name , password and IP, etc.)
        - Initialize the required VM for deployment of Docker Container **(only for a single service : Life Cycle Manager)**. Life Cycle Manager acts as the starting point of all other services.

- Life Cycle Manager runs over Azure VMs and its configuration is mostly online.
    - It contains local configs :
        - Required to login to Azure (details like, userID or subscriptionID)
        - (If required) Provisioning VMs. (details like, VM type and configuration, etc.) Configuration was hard coded locally!
        - Location of code base of all the services on Azure File Storage (i.e. source addresses)

    - Configuration through online DB includes:
        - Details for accessing VMs. (details account name , password and IP, etc.) All dynamically added Vm get into the same DB online.
        - Config contains mapping of All the service and on which VM they are required to be running i.e. (IP + port number) All maintained in a single DB to facilitate all other microservices to communicate with each other.

    - Life Cycle Manager have additional functionality to initialize Kafka on VM

On starting Bootstrapper checks if :

- VM present in the config is available or not.
- If VM is not provisioned it provisions it. After provisioning it sets up the Environment for deployment of docker containers.
- If already provisioned, it checks if it's running.
- After Running the VM, it checks if the required Docker image (of Life Cycle manager) is present on the VM or not, If not it downloads the code base from Azure File storage, makes a docker image and starts it.
- After starting the Life Cycle manager it updates the Central DB for the IP and port to let other services (mainly monitoring service) contact that IP and port.
- Life Cycle Manager acts as the starting point of all other services.

On starting Life Cycle manager checks if :

- All the VM present in the DB are available or not.
- If any VM in DB is not provisioned it provisions it. After provisioning it sets up the Environment for deployment of docker containers.
- If already provisioned, it checks if they are running, if not it starts them.
- After Running the VMs, it checks if required Docker images are present on the VM or not, If not it downloads the code base from Azure File storage, makes docker images and starts them.
- After starting each of the services it updates the Central DB for the IP and port for the started process to let other services contact that IP and port.
- It uses multithreading to simultaneously begin initialization of all the required VMs in parallel.

Note that monitoring service needs to be the last service to be started, as it will start checking as soon as it gets started. If started earlier it will needlessly send requests to the Lifecycle manager to start unstarted services.
But for that we are starting the Flask server at Lifecycle manager's end only after all the services are up and running.

Life cycle manager also contains APIs that might be required for :

- Dynamically Provisioning a VM.
- (In case if needed) Initializing any VM for Docker Environment.
- Fault Tolerance (Requires Different DB maintained for each of the below):
  - Restarting any VM
    - If a VM is restarted all the Services running on it need to get restarted as well.
    - But the above need not be handled by us as Azure VM are stateful so if a VM is restarted all other processes are automatically restarted.
  - Restarting any Microservices
  - Restarting any hosted application
  - Restarting any hosted AI Model.

Future Scope for Life Cycle Manager:

- Multiple containers of same microservice
- Dynamically initialization of multiple containers for the same service.
- Shifting of services from one node to another during run time.

Future Scope for Bootsrapper:

- Providing a central UI to configure online DB that is required by Life Cycle Manager and other services.

Shortcomings:

- What happens if the Lifecycle manager itself stops? Not handled
- What Happens if the monitoring service itself stops? Not Handled.
- Above functions of Bootstrapper and Lifecycle manager are what ideally what should have happened, but in our code there is one difference that, It's the Bootstrapper that 1st starts all the VM present in Online DB, Lifecycle Manager don’t starts/provision VM when started, it's just initializes the environment and sets up docker images.
- Lifecycle manager does Provisions VM when a request comes in. (i.e. Horizontal Scaling)