Viam Solutions Engineer Interview Exercise: Notes

Notes (to be filled out as I complete the tasks):

(screenshots are in **bold**)

Task 1: Vision

From the interview document:
"Using a camera, this could be the one on your laptop, or any USB camera. We would like to see you *configure a basic camera component in Viam* and *expand on it to make use of our vision service to implement a simple object detector.*"

Prereqs:

Hardware requirements

- Computer with a webcam (MacBook Pro)

Software requirements

- Python 3.8 or newer (I have Python 3.12.8)
- viam-server
- Viam Python SDK
    - The Viam Python SDK (software development kit) lets you control your Viam-powered machine by writing custom scripts in the Python programming language. Install the Viam Python SDK by following these instructions.

Helpful tutorial: https://docs.viam.com/tutorials/projects/send-security-photo/
Helpful Youtube Video: https://www.youtube.com/watch?v=Py6xzRA0-vg

Steps:

1) Create a Viam account. (**task1_account_created**)
2) Add a new machine using the button in the top right corner of the LOCATIONS tab in the app. A machine represents your device.
3) On the machine's page, follow the setup instructions to install viam-server on the computer you're using for your project. Wait until your machine has successfully connected to the Viam app. (**task1_machine_live**)
4) Configure your webcam so that your machine can get the video stream from your camera:
    a) Navigate to the CONFIGURE tab of your machine's page in the Viam app.

b) Click the + icon next to your machine part in the left-hand menu and select Component or service.
c) Select the camera type, then select the webcam model. Enter a name (cam) or use the suggested name for your camera and click Create.
d) Edit and fill in the attributes as applicable (JSON). Leave the video_path blank and the camera will use the default video path for your machine.

   Note: I created a Webcam discovery service to find the video_path for my computer's built-in camera (discovery_1). (**task1_find_camera_path**)

e) Click Save in the top right corner of the screen to save your changes.

5) Test your physical camera. To test your camera, go to the CONTROL tab and click to expand your camera's panel. (**task1_test_camera, task1_camera_config_JSON**)
6) Toggle View 'cam' to the "on" position. The video feed should display. If it doesn't, double-check that your config is saved correctly, and check the **LOGS** tab for errors.

7) Configure the ML Model vision service on your machine. I will use the pre-trained Machine Learning model from the Viam Registry called `EfficientDet-COCO`.
   (**task1_mlmodel_efficient_det-COCO, task1_vision_config_mlmodel_JSON**)
   a) The model can detect a variety of things, including `Persons`. You can see a full list of what the model can detect in `labels.txt` file.
   b) Navigate to your machine's CONFIGURE tab.
   c) Click the + (Create) button next to your main part in the left-hand menu and select Component or service.
   d) Start typing ML model and select ML model / TFLite CPU from the builtin options.
   e) Enter people as the name, click Add Module, then click Create.
   f) Select Deploy model on machine for the Deployment field.
   g) Then select the viam-labs:EfficientDet-COCO model from the Select model dropdown.
8) Configure an 'mlmodel detector' vision service. (**task1_myPeopleDetector, task1_myPeopleDetector_JSON**)
   a) Click the + (Create) button next to your main part in the left-hand menu and select Component or service. Start typing ML model and select vision / ML model from the builtin options.
   b) Enter 'myPeopleDetector' as the name, then click Create.
   c) In the new vision service panel, configure your service.
      i) Select 'people' from the ML Model dropdown.
      ii) Click Save in the top right corner of the screen.

9) To test that the vision service is working, click on the Test area of the vision service panel myPeopleDetector or click on the vision service panel on the CONTROL tab.
   a) If the vision service detects a person on the configured camera, you will see a red box around the detection along with the confidence score of the detection.

   Note: Got a connection error regarding my webcam here. Tried restarting the whole david-macbook-main machine. Re-ran the following to re-start viam-server.
   (**task1_connection_error**)

```
viam-server -config ~/Downloads/viam-david-macbook-main.json
```

Camera reconnected, and the test was successful. (**task1_detector_test**)

10) To leverage the vision service to operate using code, I will first install the Viam SDK. I installed via command line (**test1_viam-python-sdk_CLI_install**). I made sure to install the following to have the right dependencies (numpy).

```
pip install 'viam-sdk[mlmodel]'
```

11) Write Python code to call the vision service and detect people within the camera view (vision_service.py)
    a) I first used 'get_detections_from_camera' to see the detected objects and the model's confidence (**task1_all_objects_detected**)
    b) Then, I filtered through the detected objects list to only detect people and list the confidence (for people detected with confidence greater than 0.5) (**task1_people_only_detected, task1_people_only_detected_python**)
12) Screenshot output from command window (people detected, confidence)

Task 2: Cloud Integration

From the interview document: "Now that we have configured our vision service, we would like you to configure our data capture service to store camera frames into the cloud. How could we use this data to possibly create a custom model?"

Prereqs: Read https://docs.viam.com/data-ai/capture-data/capture-sync/.

1) Enable the data management service for your machine by clicking '+' next to the machine's name, and entering data management. Name the service data-manager.
2) Navigate to a configured resource in Builder mode in the Viam app.
3) Find the Data capture section in the resource panel.
4) Click '+ Add Method'.
5) If you see a warning Capture disabled on data management service, click Enable capture on data management service. (**task2_data_management_service warning, task2_data_manager_enabled**)
6) Select a **Method** to capture data from.
7) Set the capture frequency in hertz, for example to 0.2 with ReadImage on a camera to capture an image every 5 seconds. (**task2_data_manager_configured_with_camera**)
8) Save your config.
9) Click the 'Data' tab and view the images saved to the cloud. (**task2_images_stored_to_the_cloud**)
10) Export the data using the CLI. (**task2_viam_CLI_data_export**)
11) Enable Data Capture method for 'myPeopleDetector' vision service. (**task2_myPeopleDetector_DataCapture**)

12) View images with detection metadata in the 'Data' tab. (**task2_vision_service_cloud_data**)

Note: you could set up data capture and sync configuration via JSON (https://docs.viam.com/data-ai/capture-data/advanced/advanced-data-capture-sync/).

Task 3: Modular Registry

When working with customers, we often come across a wide variety of hardware components and systems that we have to integrate the Viam platform with. The Viam Registry allows managing these custom integrations as so-called modules. The Registry is comparable to Node.js npm or Python pip, just for what we call smart machines. Now that we have our object detection working, we would like you to create a custom sensor which will call the previously created vision service and return true if an object is detected or false if no object is detected. This will depend on the service created above with a single field called "person detected" and it is set to 1 if our detection has a person and 0 if no person is detected.The goal of this exercise is to later show and explain how simple and fast you / our customers can do this with the Viam platform.

Prereqs/guidance: https://codelabs.viam.com/guide/control-module/index.html#0
From docs: https://docs.viam.com/operate/get-started/other-hardware/#upload-your-module
Hello world module: https://docs.viam.com/operate/get-started/other-hardware/hello-world-module/
Helpful Youtube video: https://www.youtube.com/watch?v=TnOuch70mnI

Steps:

1) Make sure the viam-module CLI tool is installed.
2) Scaffold a New Custom Sensor Module.
    a) Enter the following into the terminal (after navigating to the project's directory): `viam login`, `viam module generate` **(task3_module_generate)**
    b) When prompted, enter: Set a module name: `sensor-pd`
    c) When prompted, specify the language for the module: `Python`
    d) When prompted, specify the visibility: `Private`
    e) When prompted, enter the Namespace/Org ID: `dl-org`
    f) When prompted, enter: What type of resource are you creating? `Sensor Component`
    g) When prompted, enter: What will the model name be? `pdetect`
    h) When prompted, enter: Enable cloud build: `Yes`
    i) When prompted, enter: Register module: `Yes`
    j) Accept the default viam-labs namespace. This creates the module's folder structure within your project directory. (**task3_module_created_ack**)
3) Implement sensor logic in src/viam_labs/sensor/pdetect.py and replace the default logic.
    a) Edit the validate_config function to do the following: (**task3_validate_config()**)
        i) Check that the user has configured required attributes and return errors if they are missing.
        ii) Return a map of any dependencies.
    b) Edit the reconfigure function, which gets called when the user changes the configuration. This function should do the following: (**task3_reconfigure()**)
        i) If you assigned any configuration attributes to global variables, get the values from the latest config object and update the values of the global variables.

ii) Assign default values as necessary to any optional attributes if the user hasn't configured them.
iii) If your module has dependencies, get the dependencies from the dependencies map and cast each resource according to which API it implements, as in this ackermann.py example.

c) Edit the methods you want to implement: (**task3_do_command()**)
   i) For each method you want to implement, replace the body of the method with your relevant logic. Make sure you return the correct type in accordance with the function's return signature. You can find details about the return types at python.viam.dev.

d) Add logging messages as desired. The following log severity levels are available for resource logs (self.logger. – debug, info, warn, error, exception, critical)

e) Edit the generated requirements.txt file to include any packages that must be installed for the module to run. Depending on your use case, you may not need to add anything here beyond viam-sdk which is auto-populated.

Note:

- Added logic to call myPeopleDetector and return structured result within do_command(()
- Imported VisionClient to call the vision service
- validate_config() returns ["detector_name"], [] (required dependency)
- reconfigure() injects self.vision from dependencies
- get_readings() calls do_command() doesn't wrap return in SensorReading
- Maintained Docstrings
- Declared instance variables within __init__ for clarity
- Needed to edit the new() method to return the sensor instance itself (**task3_new()**)
- I needed to update main.py to register the module, and I then I kept getting 'API "rdk:component:sensor" with model "dl-org:sensor-pdr:pdetect" not registered' because my module was crashing due to duplicate registration
  - Added a try:catch block to prevent this. (**task3_notRegisteredError**)
- Then I got a 'ConnectError: [unknown] TypeError - Cannot instantiate typing.Union - file_name='/Users/davidlevine/miniconda3/lib/python3.12/typing.py' func_name='__call__' line_num=501'
  - Fixed by removing SensorReading() (**task3_typeError**)

4) Prepare to run your module

a) Hot reloading – create a reload.sh script in your module directory. You'll use this for local testing and can delete it before you upload your module. Paste the following contents into it and save the file: (**task3_reload_sh**)

```
#!/usr/bin/env bash

# bash safe mode. look at `set --help` to see what these are doing
set -euxo pipefail

cd $(dirname $0)
MODULE_DIR=$(dirname $0)
VIRTUAL_ENV=$MODULE_DIR/venv
PYTHON=$VIRTUAL_ENV/bin/python
./setup.sh
```

```
# Be sure to use `exec` so that termination signals reach the
python process,
# or handle forwarding termination signals manually
exec $PYTHON src/main.py $@
```

b) Make your reload script executable by running the following command in your module directory: `chmod 755 reload.sh`
c) Create a virtual Python environment with the necessary packages by running the setup file from within the module directory (**task3_setup_sh**): `sh setup.sh`
d) Edit your meta.json, replacing the "entrypoint", "build", and "path" fields as follows (**task3_meta_json**):

```
"entrypoint": "reload.sh",
"first_run": "",
"build": {
        "build": "rm -f module.tar.gz && tar czf module.tar.gz requirements.txt src/*.py
src/models/*.py meta.json setup.sh reload.sh",
        "setup": "./setup.sh",
        "path": "module.tar.gz",
        "arch": [
                "linux/amd64",
                "linux/arm64",
                "darwin/arm64"
                ]
        }
```

5) Configure your local module on a machine
    a) Hot reloading – Run the following command to build and start your module and push it to your machine (**task3_viam_module_rebuild**): `viam module reload --local`

    Needed: `viam module reload --local --part-id accf7c3d-370b-4268-a869-70d878f3e61a`
    b) Module 'sensor-pd by dl-org' should appear automatically (**task3_sensor-pd**)

6) Configure the model provided by your module (**task3_add_resource**)
    a) Click the + button again, this time selecting Local module and then Local component.
    b) Select or enter the model namespace triplet, for example dl-org:sensor-pd:pdetect. You can find the triplet in the model field of your meta.json file.
    c) Select the Type corresponding to the API you implemented.
    d) Enter a Name such as sensor-1. Click Create.
    e) Configure any required attributes using proper JSON syntax. (needed "camera_name":"cam", "detector_name": "myPeopleDetector") (**task3_sensor_pd_test**)

    Note: once steps 1-5 were complete, I could click 'Restart module' within the Viam UI at sensor-pd to refresh changes to the code – didn't need to re-run steps 1-5. Viewed logs to see errors.

7) Test the sensor: once added, go to Control > pdetect (**task3_person_detector_test**) (**video**)
    a) Click Test > GetReadings > should return {“person_detected”: 1} or 0