VIAM

Solutions Engineer Interview Exercise

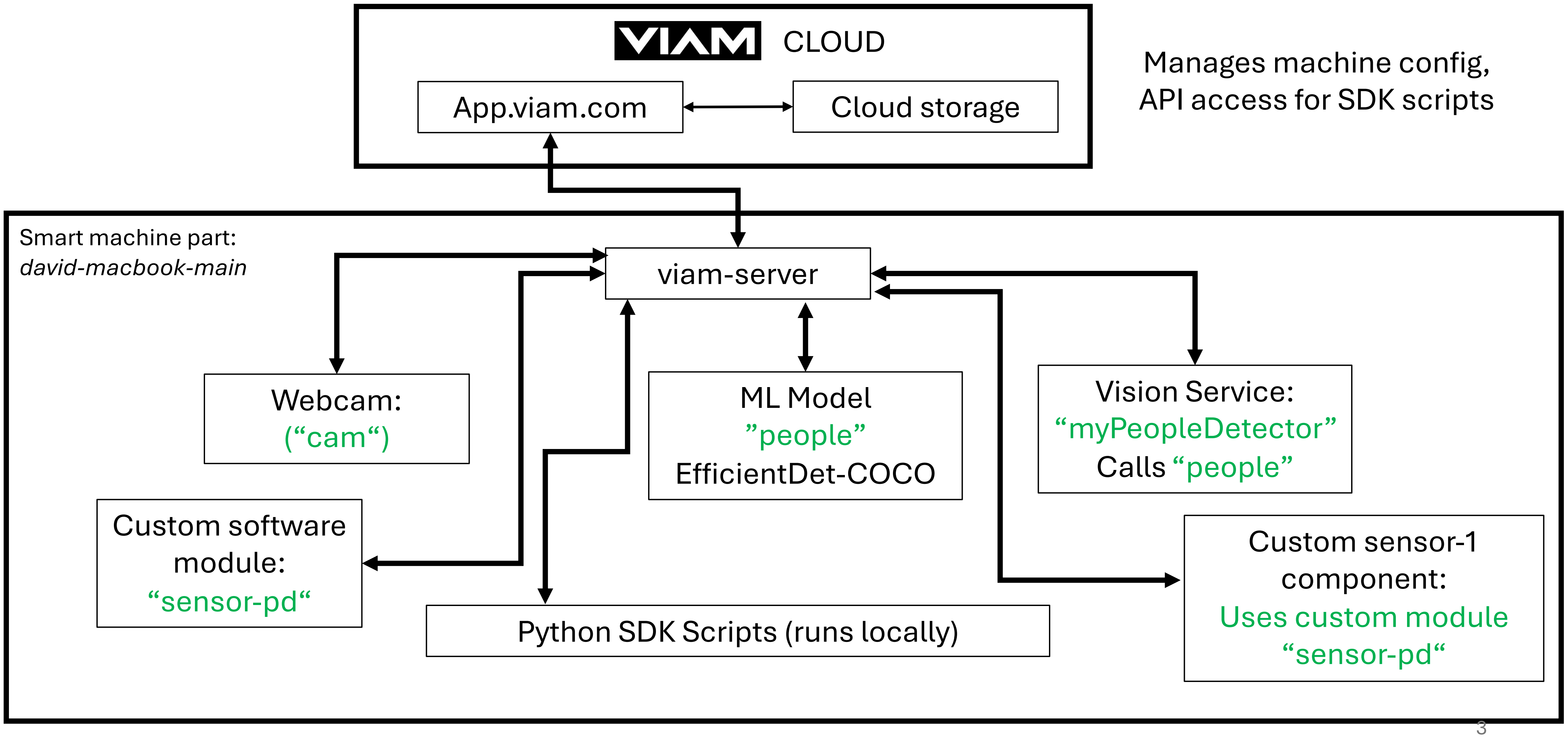
David J. Levine

Solutions Engineer Candidate

Overview of the Project

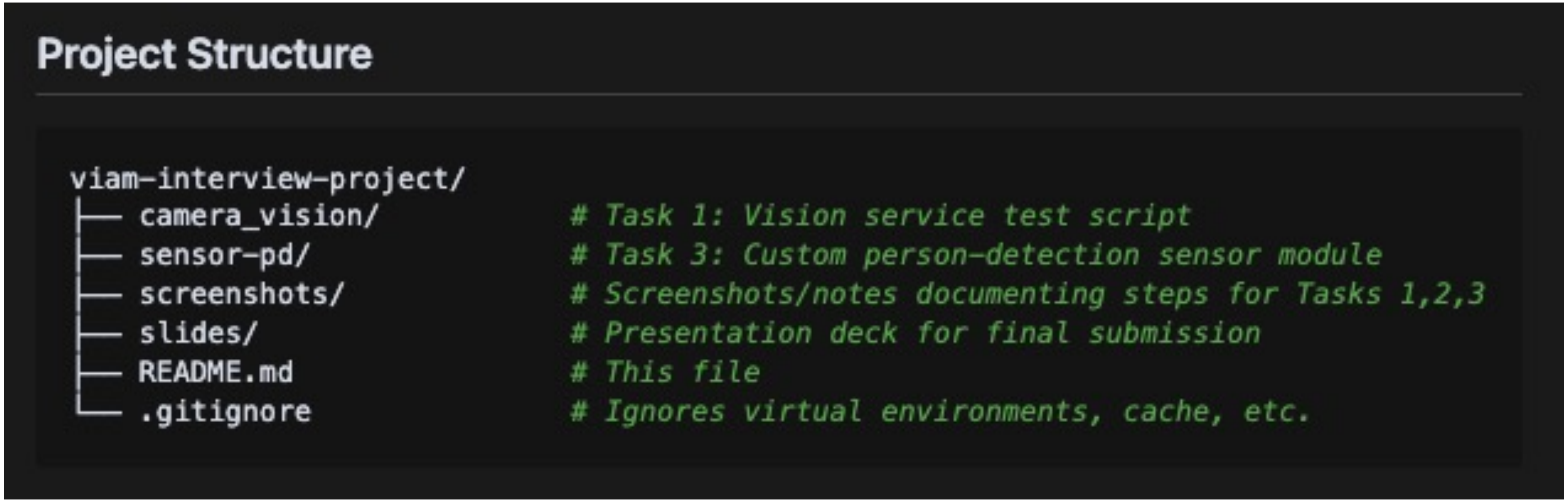
- Task 1: Vision
  - Configure a basic camera component and use Viam’s vision service to implement a simple object detector
- Task 2: Cloud Integration
  - Configure Viam’s data capture service to store camera frames into the cloud
  - How could we use this data to possibly create a custom model?
- Task 3: Modular Registry
  - Create a custom sensor (calling the previously created vision service)
  - Should have a single field called “person detected”
  - Set to 1 if a person is detected, and 0 if no person is detected

Platform architecture



Manages machine config, API access for SDK scripts

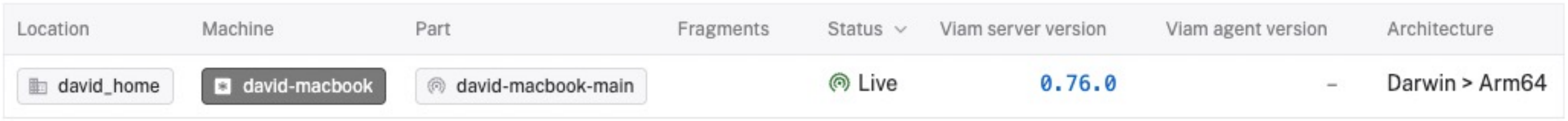
Overview of the Project Directory



Project Structure

```
viam-interview-project/
├── camera_vision/          # Task 1: Vision service test script
├── sensor-pd/              # Task 3: Custom person-detection sensor module
├── screenshots/            # Screenshots/notes documenting steps for Tasks 1,2,3
├── slides/                 # Presentation deck for final submission
├── README.md               # This file
└── .gitignore              # Ignores virtual environments, cache, etc.
```

From GitHub README: https://github.com/djlev8/viam-interview-project/tree/main



Location	Machine	Part	Fragments	Status	Viam server version	Viam agent version	Architecture
david_home	david-macbook	david-macbook-main		Live	0.76.0	–	Darwin > Arm64

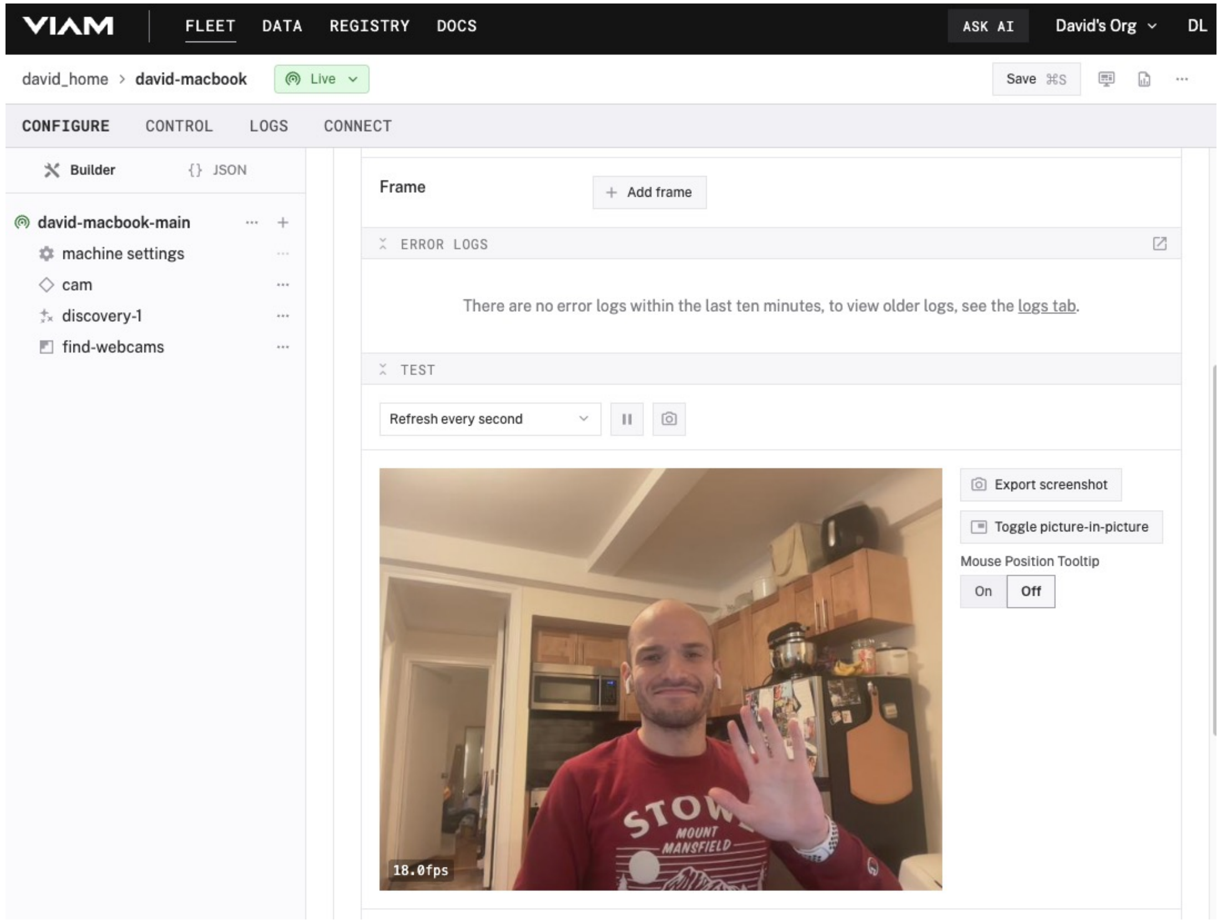
From Viam dashboard

Task 1: Vision Service and Object Detection

Following the ‘Detect a Person and Send a Photo’ tutorial.

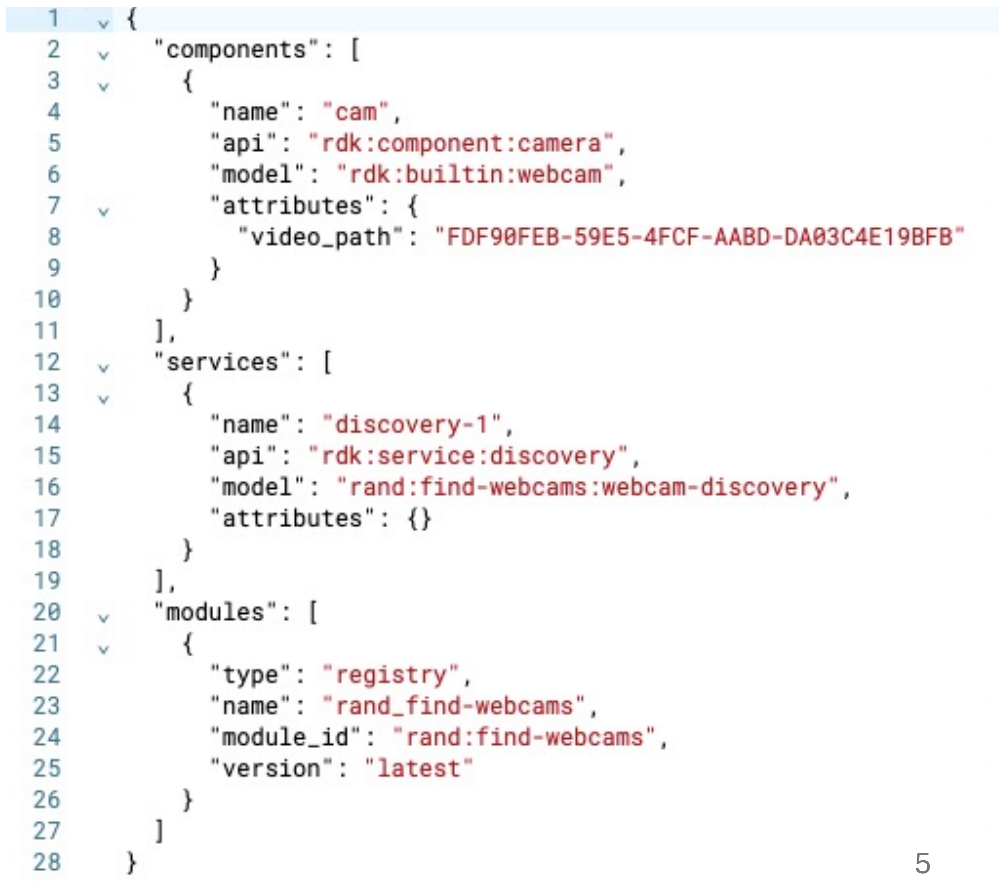
Goal: **Set up a webcam** → ML model (EfficientDet-COCO) → Vision Service → SDK-based person detection

Configured webcam test



There are no error logs within the last ten minutes, to view older logs, see the logs tab.

Configured webcam JSON

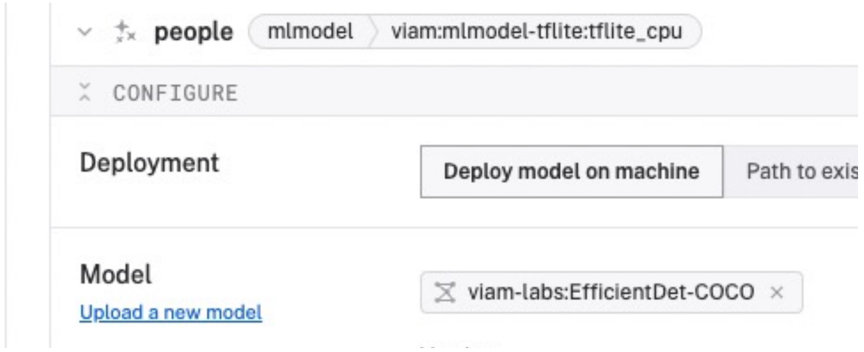


```json
{
  "components": [
    {
      "name": "cam",
      "api": "rdk:component:camera",
      "model": "rdk:builtin:webcam",
      "attributes": {
        "video_path": "FDF90FEB-59E5-4FCF-AABD-DA03C4E19BFB"
      }
    }
  ],
  "services": [
    {
      "name": "discovery-1",
      "api": "rdk:service:discovery",
      "model": "rand:find-webcams:webcam-discovery",
      "attributes": {}
    }
  ],
  "modules": [
    {
      "type": "registry",
      "name": "rand_find-webcams",
      "module_id": "rand:find-webcams",
      "version": "latest"
    }
  ]
}
```

Task 1: Vision Service and Object Detection

Goal: Set up a webcam → **ML model (EfficientDet-COCO)** → **Vision Service** → SDK-based person detection

EfficientDet-COCO model deployed



people | mlmodel | viam:mlmodel-tflite:tflite_cpu

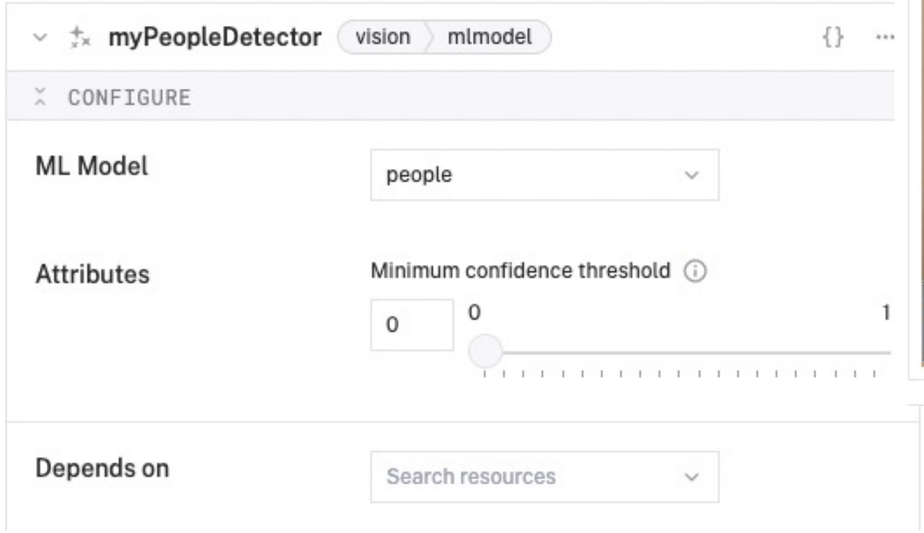
CONFIGURE

Deployment: Deploy model on machine | Path to exis

Model: viam-labs:EfficientDet-COCO

Upload a new model

Vision Service created and linked to model



myPeopleDetector | vision | mlmodel

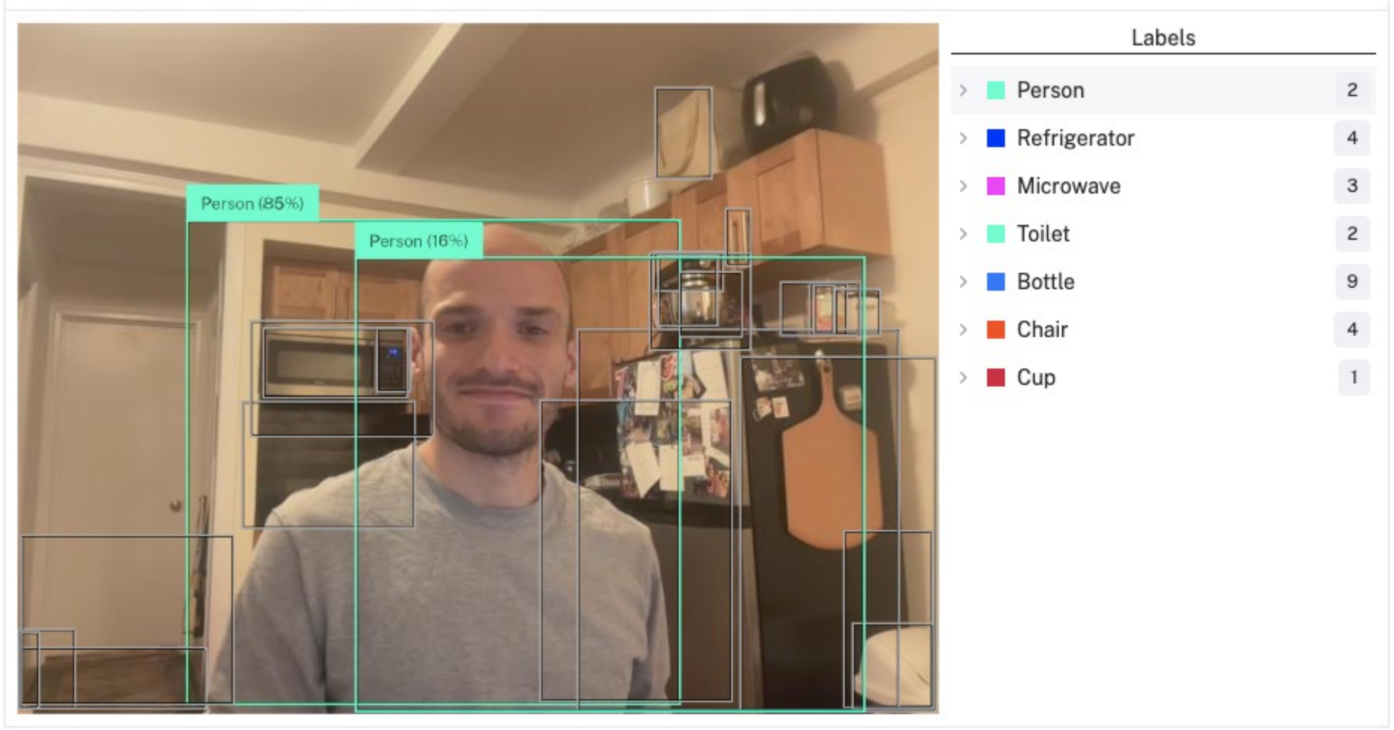
CONFIGURE

ML Model: people

Attributes: Minimum confidence threshold: 0 (slider 0–1)

Depends on: Search resources

Detection UI

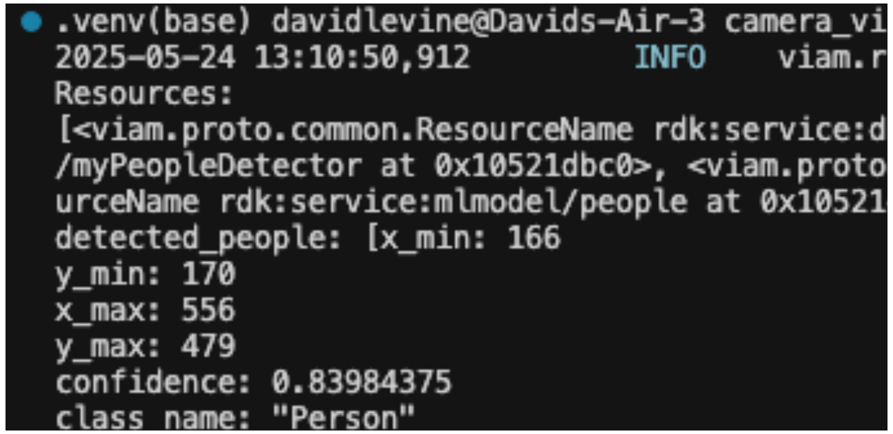


Labels	
🟩 Person	2
🟦 Refrigerator	4
🟪 Microwave	3
🟩 Toilet	2
🟦 Bottle	9
🟧 Chair	4
🟥 Cup	1

Task 1: Vision Service and Object Detection

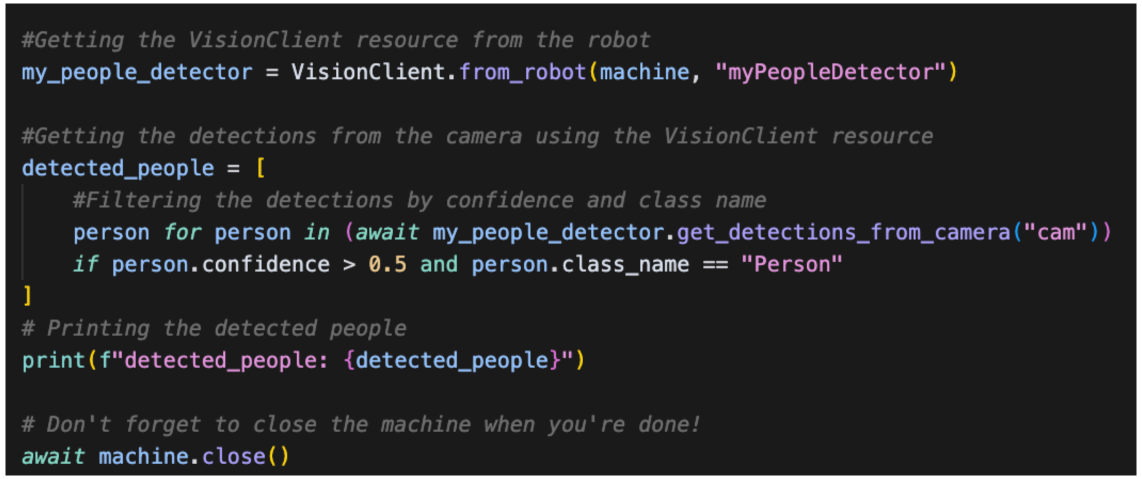
Goal: Set up a webcam → ML model (EfficientDet-COCO) → Vision Service → **SDK-based person detection**

Terminal Output: Person Detected



```
● .venv(base) davidlevine@Davids-Air-3 camera_vi
  2025-05-24 13:10:50,912        INFO    viam.r
  Resources:
  [<viam.proto.common.ResourceName rdk:service:d
  /myPeopleDetector at 0x10521dbc0>, <viam.proto
  urceName rdk:service:mlmodel/people at 0x10521
  detected_people: [x_min: 166
  y_min: 170
  x_max: 556
  y_max: 479
  confidence: 0.83984375
  class_name: "Person"
```

Code snippet from detection script (vision_service.py)



```python
#Getting the VisionClient resource from the robot
my_people_detector = VisionClient.from_robot(machine, "myPeopleDetector")

#Getting the detections from the camera using the VisionClient resource
detected_people = [
    #Filtering the detections by confidence and class name
    person for person in (await my_people_detector.get_detections_from_camera("cam"))
    if person.confidence > 0.5 and person.class_name == "Person"
]
# Printing the detected people
print(f"detected_people: {detected_people}")

# Don't forget to close the machine when you're done!
await machine.close()
```

Used SDK to filter for confident (> 0.5) 'Person' detections, instead of printing all detections found, regardless of class name or confidence

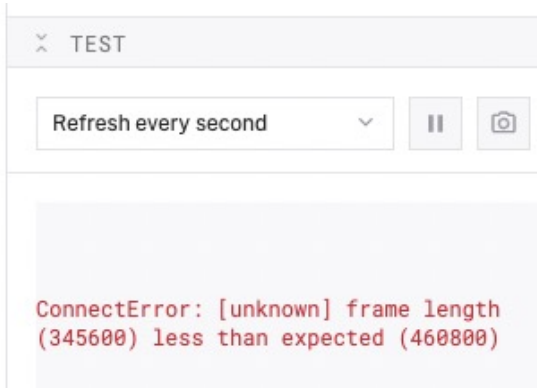
Task 1: Vision Service and Object Detection

Challenges & Debugging:

- Encountered webcam connection error
    - Restarted viam-server and reconnected successfully
    - CLI: `viam-server -config ~/Downloads/viam-david-macbook-main.json`

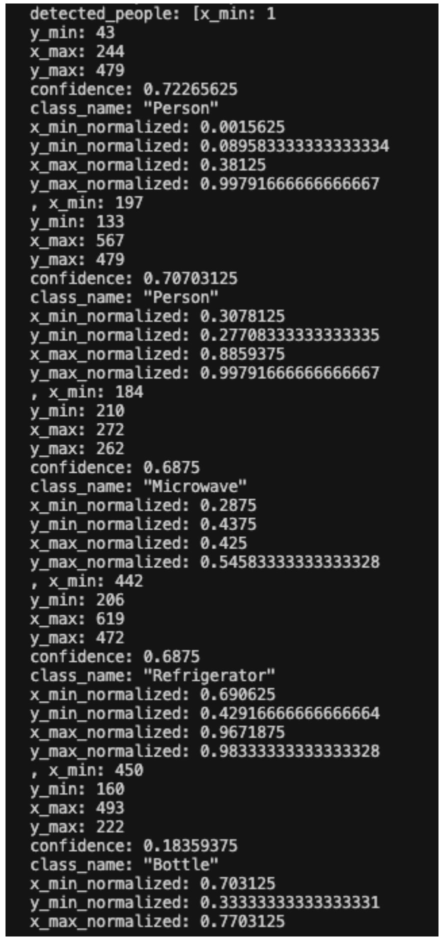
- Needed to use discovery service to retrieve valid video path for my built-in webcam

- Filtered through list of detections to find confident 'person' only detections after first obtaining all detections



```
ConnectError: [unknown] frame length
(345600) less than expected (460800)
```

Webcam connect error



```
detected_people: [x_min: 1
y_min: 43
x_max: 244
y_max: 479
confidence: 0.72265625
class_name: "Person"
x_min_normalized: 0.0015625
y_min_normalized: 0.089583333333333334
x_max_normalized: 0.38125
y_max_normalized: 0.99791666666666667
, x_min: 197
y_min: 133
x_max: 567
y_max: 479
confidence: 0.70703125
class_name: "Person"
x_min_normalized: 0.3078125
y_min_normalized: 0.27708333333333335
x_max_normalized: 0.8859375
y_max_normalized: 0.99791666666666667
, x_min: 184
y_min: 210
x_max: 272
y_max: 262
confidence: 0.6875
class_name: "Microwave"
x_min_normalized: 0.2875
y_min_normalized: 0.4375
x_max_normalized: 0.425
y_max_normalized: 0.54583333333333328
, x_min: 442
y_min: 206
x_max: 619
y_max: 472
confidence: 0.6875
class_name: "Refrigerator"
x_min_normalized: 0.690625
y_min_normalized: 0.42916666666666664
x_max_normalized: 0.9671875
y_max_normalized: 0.98333333333333328
, x_min: 450
y_min: 160
x_max: 493
y_max: 222
confidence: 0.18359375
class_name: "Bottle"
x_min_normalized: 0.703125
y_min_normalized: 0.33333333333333331
x_max_normalized: 0.7703125
```

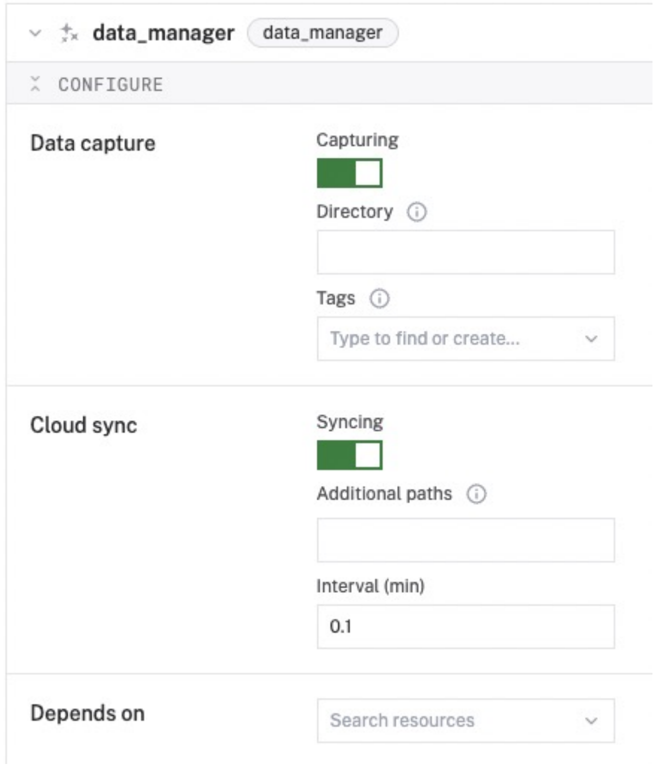
Initial output: Detecting many objects

Task 2: Data Capture & Cloud Storage

Following the 'Capture and sync edge data' document.

Goal: **Create data-manager service → Set capture rate & ReadImage method** → Store frames to cloud

Data management service enabled



data_manager data_manager

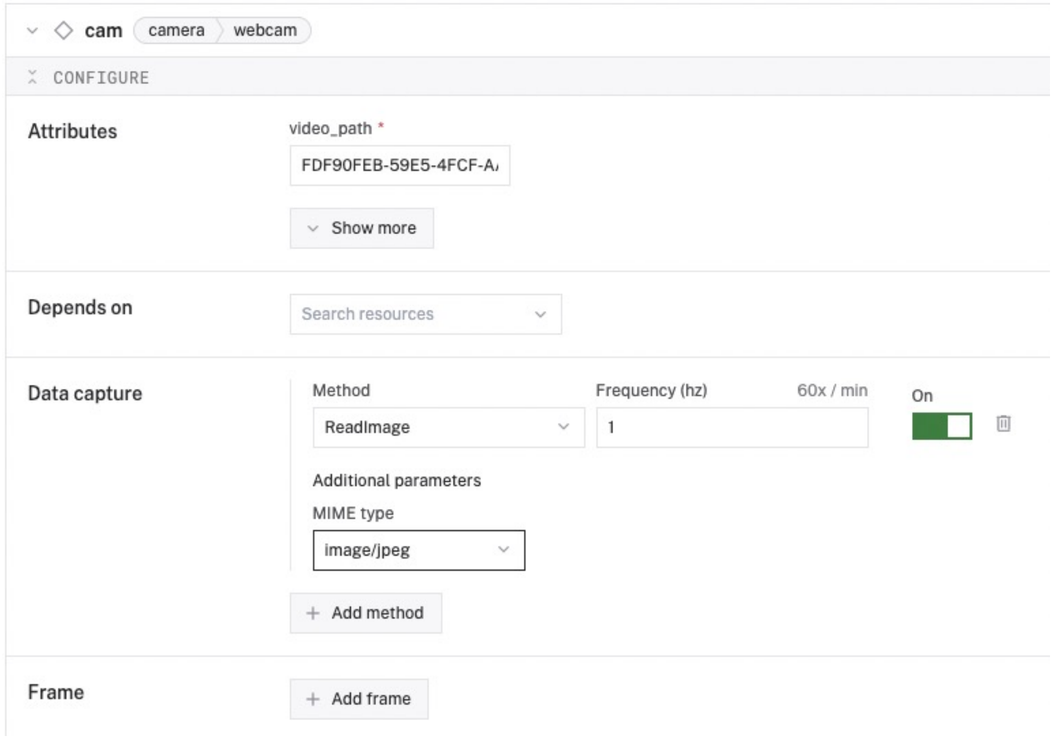
CONFIGURE

Data capture
- Capturing: On
- Directory:
- Tags: Type to find or create...

Cloud sync
- Syncing: On
- Additional paths:
- Interval (min): 0.1

Depends on: Search resources

Data capture settings specified (capture rate = 1 Hz, MIME type = JPG)



cam camera webcam

CONFIGURE

Attributes
- video_path *: FDF90FEB-59E5-4FCF-A
- Show more

Depends on: Search resources

Data capture
- Method: ReadImage
- Frequency (hz): 1 (60x / min)
- On: On
- Additional parameters
- MIME type: image/jpeg
- Add method

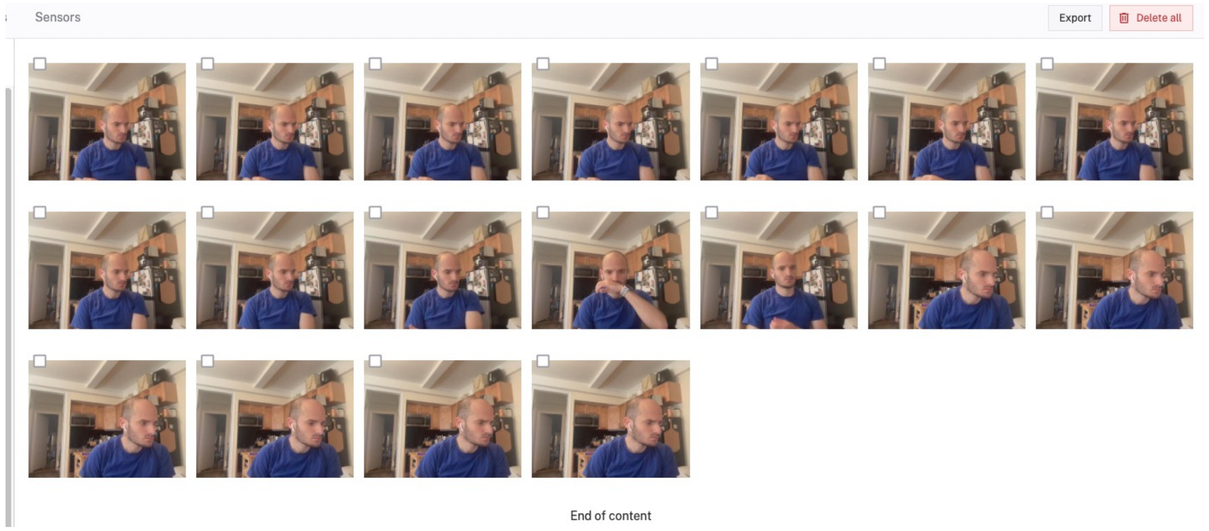
Frame: Add frame

Note: Needed to create data management service prior to configuring data capture with configured camera.

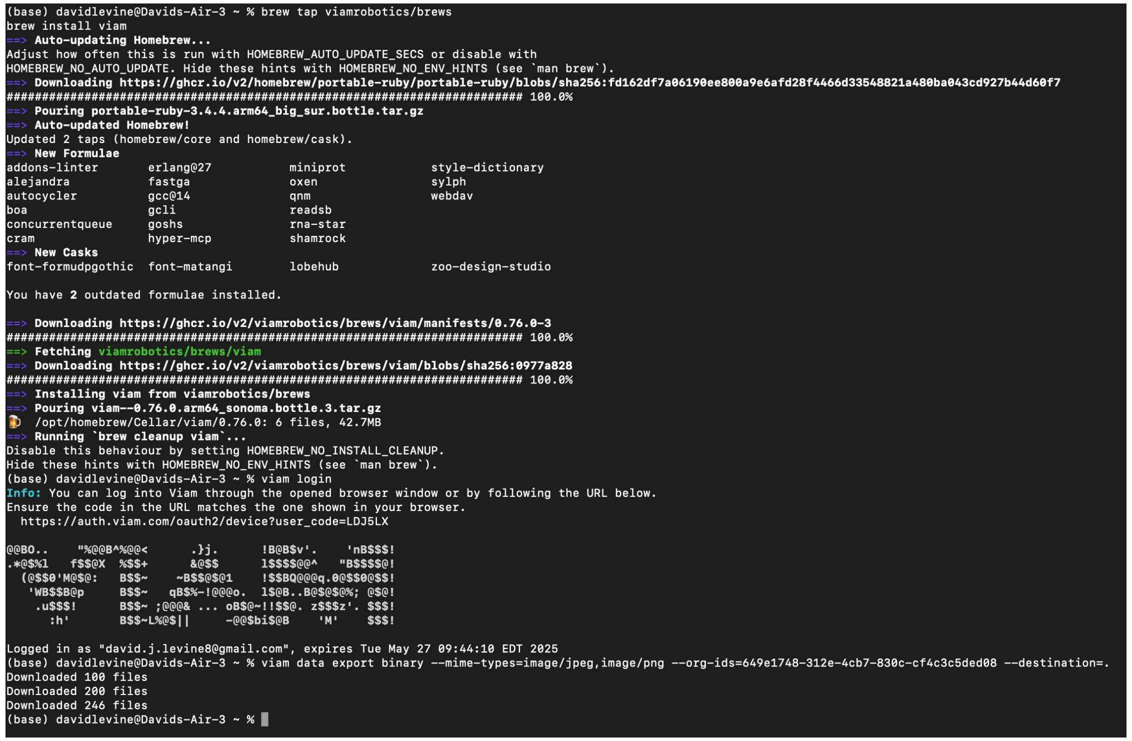
Task 2: Data Capture & Cloud Storage

Goal: Create data-manager service → Set capture rate & ReadImage method → **Store frames to cloud**

Images stored to the cloud



Installed viam CLI to run image export



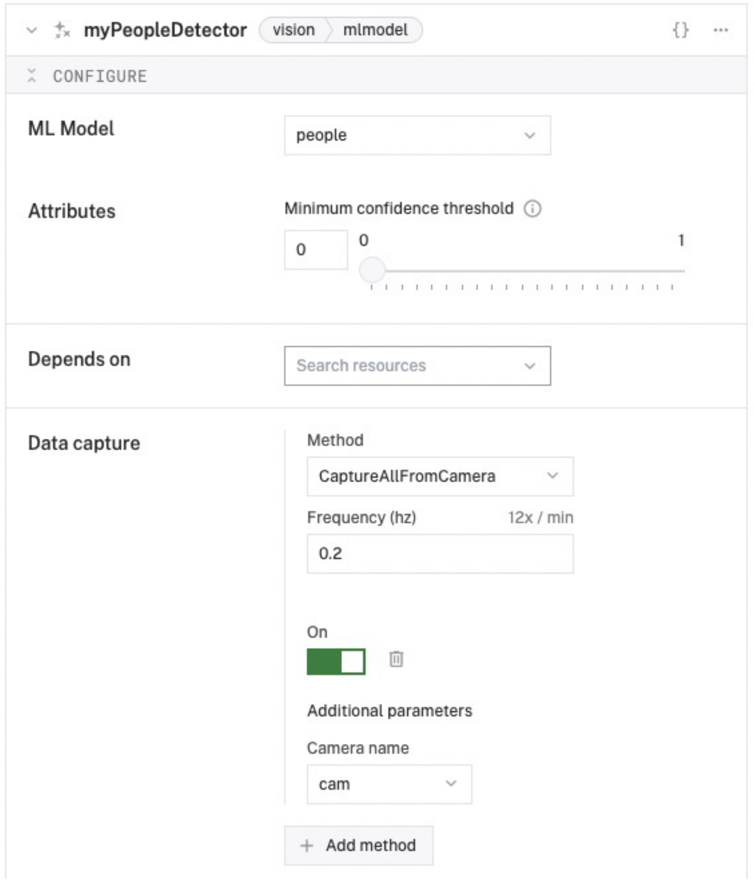
```
(base) davidlevine@Davids-Air-3 ~ % brew tap viamrobotics/brews
brew install viam
==> Auto-updating Homebrew...
Adjust how often this is run with HOMEBREW_AUTO_UPDATE_SECS or disable with
HOMEBREW_NO_AUTO_UPDATE. Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
==> Downloading https://ghcr.io/v2/homebrew/portable-ruby/portable-ruby/blobs/sha256:fd162df7a06190ee800a9e6afd28f4466d33548821a480ba043cd927b44d60f7
########################################################################## 100.0%
==> Pouring portable-ruby-3.4.4.arm64_big_sur.bottle.tar.gz
==> Auto-updated Homebrew!
Updated 2 taps (homebrew/core and homebrew/cask).
==> New Formulae
addons-linter       erlang@27           miniprot            style-dictionary
alejandra           fastga              oxen                sylph
autocycler          gcc@14              qnm                 webdav
boa                 gcli                readsb
concurrentqueue     goshs               rna-star
cram                hyper-mcp           shamrock
==> New Casks
font-formudpgothic  font-matangi        lobehub             zoo-design-studio

You have 2 outdated formulae installed.

==> Downloading https://ghcr.io/v2/viamrobotics/brews/viam/manifests/0.76.0-3
########################################################################## 100.0%
==> Fetching viamrobotics/brews/viam
==> Downloading https://ghcr.io/v2/viamrobotics/brews/viam/blobs/sha256:0977a828
########################################################################## 100.0%
==> Installing viam from viamrobotics/brews
==> Pouring viam--0.76.0.arm64_sonoma.bottle.3.tar.gz
🍺  /opt/homebrew/Cellar/viam/0.76.0: 6 files, 42.7MB
==> Running `brew cleanup viam`...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
(base) davidlevine@Davids-Air-3 ~ % viam login
Info: You can log into Viam through the opened browser window or by following the URL below.
Ensure the code in the URL matches the one shown in your browser.
  https://auth.viam.com/oauth2/device?user_code=LDJ5LX

@@BO..    "%@@B^%@@<      .}j.      !B@B$v'.    'nB$$$!
.*@$%l   f$$@X  %$$+      &@$$      l$$$$@@^   "B$$$$@!
  (@$$0'M@$@:   B$$~    ~B$$@$@1    !$$BQ@@@q.0@$$0@$$!
   'WB$$B@p     B$$~   qB$%-!@@@o.  l$@B..B@$@$@%; @$@!
    .u$$$!      B$$~ ;@@@& ... oB$@~!!$$@. z$$$z'. $$$!
      :h'       B$$~L%@$||     -@@$bi$@B    'M'    $$$!

Logged in as "david.j.levine8@gmail.com", expires Tue May 27 09:44:10 EDT 2025
(base) davidlevine@Davids-Air-3 ~ % viam data export binary --mime-types=image/jpeg,image/png --org-ids=649e1748-312e-4cb7-830c-cf4c3c5ded08 --destination=.
Downloaded 100 files
Downloaded 200 files
Downloaded 246 files
(base) davidlevine@Davids-Air-3 ~ %
```

```
viam data export --robot-id <robot-id> --component-name cam --method
ReadImage --output ./exported_data
```

Task 2: Data Capture & Cloud Storage

Goal: Create data-manager service → Set capture rate & ReadImage method → **Store frames to cloud**

Images stored to the cloud with detection details from myPeopleDetector Vision service



myPeopleDetector vision mlmodel

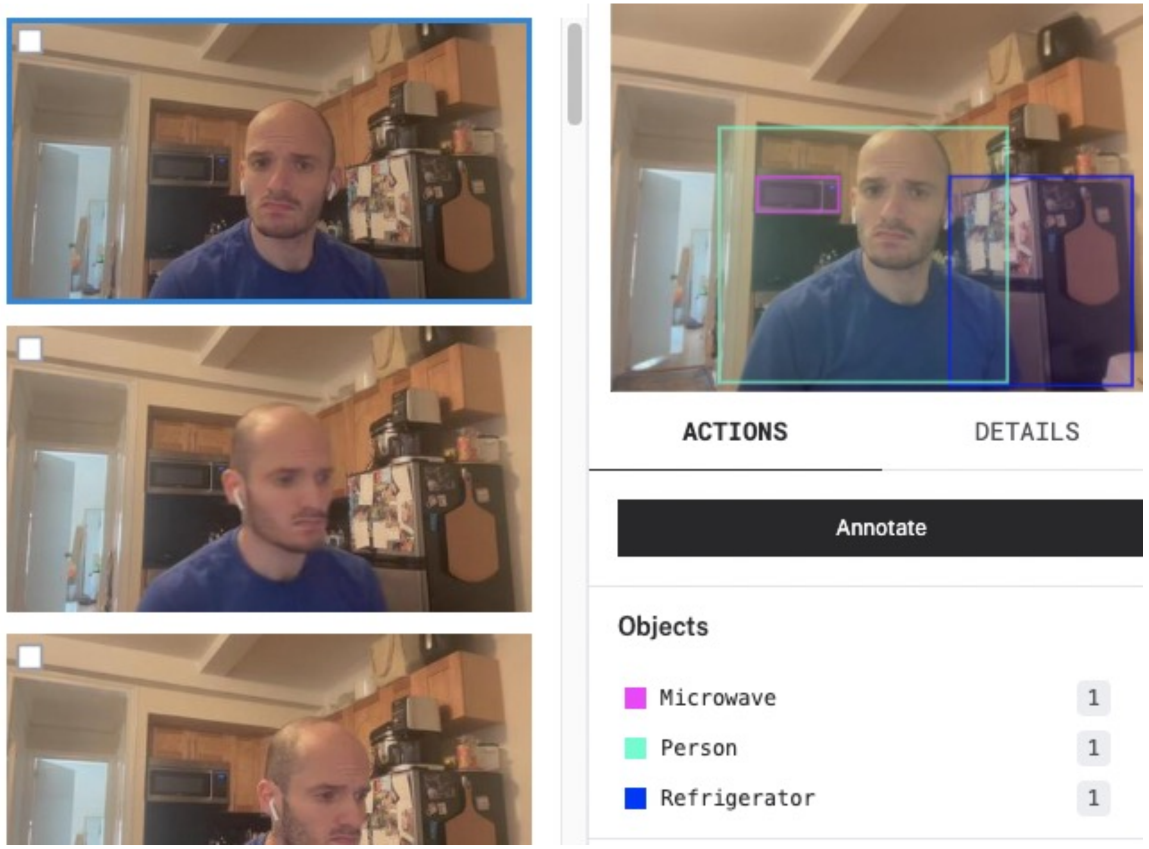
CONFIGURE

ML Model: people

Attributes: Minimum confidence threshold 0 (0 – 1)

Depends on: Search resources

Data capture
- Method: CaptureAllFromCamera
- Frequency (hz): 0.2 (12x / min)
- On
- Additional parameters
- Camera name: cam



Objects

🟪	Microwave	1
🟩	Person	1
🟦	Refrigerator	1

Task 2: Discussion – How the data enables custom models

I enabled cloud logging for the cam component, which allowed me to capture image frames from my MacBook’s webcam directly to the Viam cloud.

This type of captured image data is valuable for:

- Evaluating model accuracy: comparing what the model detects vs what’s actually in the frame.
- Labeling custom datasets: using stored images and metadata to label people or other objects.
- Retraining or fine-tuning models: creating a customer-specific version of EfficientDet-COCO that performs better on their environment or edge cases.

Exported image data and detection logs can be used to train or validate models in frameworks like PyTorch, TensorFlow, or AutoML pipelines.

Going forward, combining image data with detection metadata (e.g., bounding boxes + class + confidence) provides a complete training set for a custom object detector.

Task 3: Modular Registry: Overview

Following the ‘Integrate other physical or virtual hardware’ document.

Goal: Build a custom sensor module that uses the Vision service I previously created and returns:

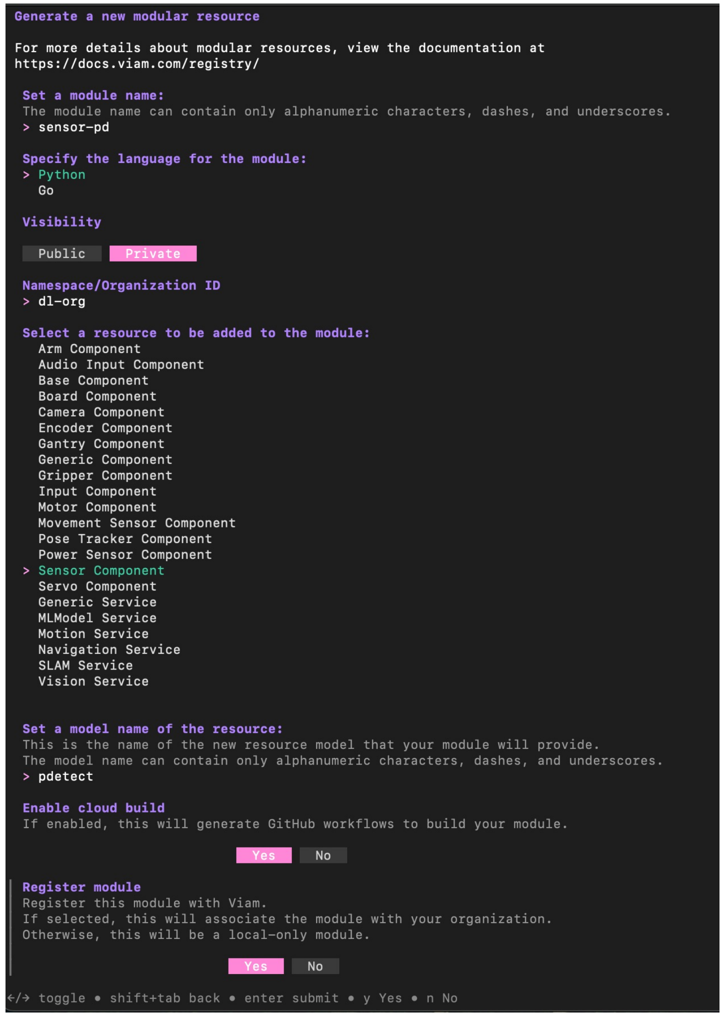
{’person_detected’: 1} if a person is visible to the camera,
and
{’person_detected’: 0} if a person is NOT visible to the camera.

Steps:

1) Module Setup
2) Implementation of the Component API
3) Adding the Custom Sensor Component to my Machine
4) Local Testing
5) Error Handling & Fixes

Task 3: Modular Registry: Module Setup

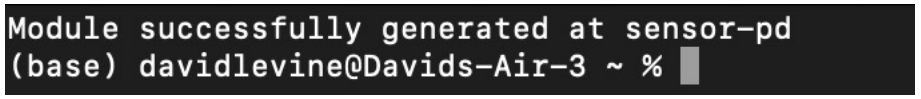
Goal: Scaffold and generate the custom module, generating the necessary stub files (in the sensor-pd folder).



```
Generate a new modular resource

For more details about modular resources, view the documentation at
https://docs.viam.com/registry/

 Set a module name:
 The module name can contain only alphanumeric characters, dashes, and underscores.
 > sensor-pd

 Specify the language for the module:
 > Python
   Go

 Visibility

   Public  [✓ Private]

 Namespace/Organization ID
 > dl-org

 Select a resource to be added to the module:
   Arm Component
   Audio Input Component
   Base Component
   Board Component
   Camera Component
   Encoder Component
   Gantry Component
   Generic Component
   Gripper Component
   Input Component
   Motor Component
   Movement Sensor Component
   Pose Tracker Component
   Power Sensor Component
 > Sensor Component
   Servo Component
   Generic Service
   MLModel Service
   Motion Service
   Navigation Service
   SLAM Service
   Vision Service


 Set a model name of the resource:
 This is the name of the new resource model that your module will provide.
 The model name can contain only alphanumeric characters, dashes, and underscores.
 > pdetect

 Enable cloud build
 If enabled, this will generate GitHub workflows to build your module.

                           [✓ Yes]  No

 Register module
 Register this module with Viam.
 If selected, this will associate the module with your organization.
 Otherwise, this will be a local-only module.

                          [✓ Yes]  No

←/→ toggle • shift+tab back • enter submit • y Yes • n No
```

From the CLI, I used `viam module generate` and followed the relevant prompts.

Module successfully generated!



```
Module successfully generated at sensor-pd
(base) davidlevine@Davids-Air-3 ~ %
```

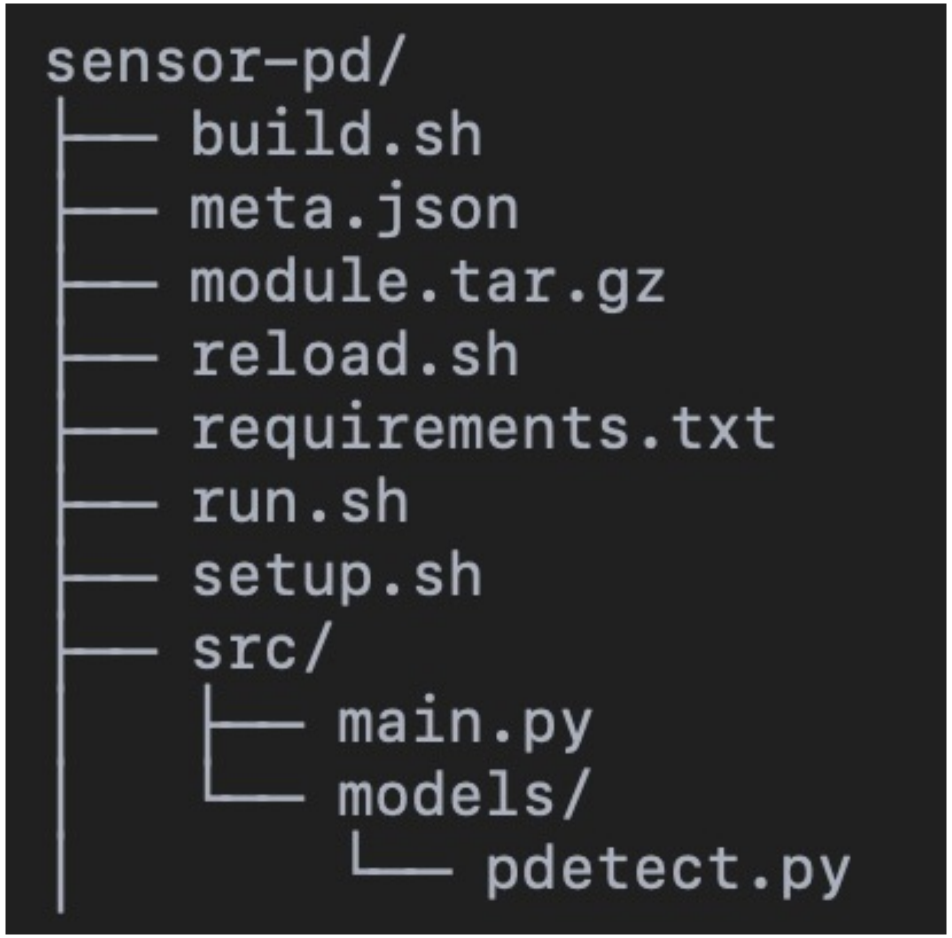
<u>Prompts filled:</u>

Name: **sensor-pd**
Language: **Python**
Visibility: **Private**
Namespace/Org-ID: **dl-org**
Resource to be added: **Sensor Component**
Model name: **pdetect**
Cloud build? **Yes**
Register module? **Yes**

Task 3: Modular Registry: Module Setup

Goal: Scaffold and generate the custom module, generating the necessary stub files (in the sensor-pd folder).

Final file structure within the generated sensor-pd folder:



```
sensor-pd/
├── build.sh
├── meta.json
├── module.tar.gz
├── reload.sh
├── requirements.txt
├── run.sh
├── setup.sh
├── src/
│   ├── main.py
│   └── models/
│       └── pdetect.py
```

I added or changed files from the auto-generated structure to implement my custom sensor logic and enable local development and testing:

- reload.sh **(I added)**
  - Script I wrote for **hot reloading** the module locally.
- meta.json **(I modified)**
- src/models/pdetect.py **(I modified)**
- src/main.py **(I modified)**

Module Architecture: How Viam Registers and Runs the Custom Sensor

- viam-server automatically launches the module when the smart machine boots.
  - For local testing, I used reload.sh to call main.py.
  - The module (your code) is registered using Module.from_args() in main.py.
  - The custom model dl-org:sensor-pd:pdetect is registered to point to the Pdetect class in pdetect.py.

- When a modular resource is created (e.g. sensor-1), the following flow happens:
  - validate_config() checks for required attributes (camera_name, detector_name) and declares dependencies.
  - new() instantiates the Pdetect class and calls reconfigure().
  - reconfigure() receives a dependency map and retrieves the Vision service using the user-supplied detector name.
  - That Vision service is stored internally to enable live detection in do_command.

- The sensor is now live. The Viam app can call get_readings() → do_command() to check if a person is detected.
- Logs and live values are visible in the Viam UI for easy debugging.

Task 3: Modular Registry: Module Implementation

Goal: Add custom logic to Pdetect class (**pdetect.py**) to execute the goals of Task 3. Following the docs & looking at the ‘simple-module’ example in the Viam Python SDK, I had to make sure I edited the following within Pdetect:

new()

validate_config()

get_readings()

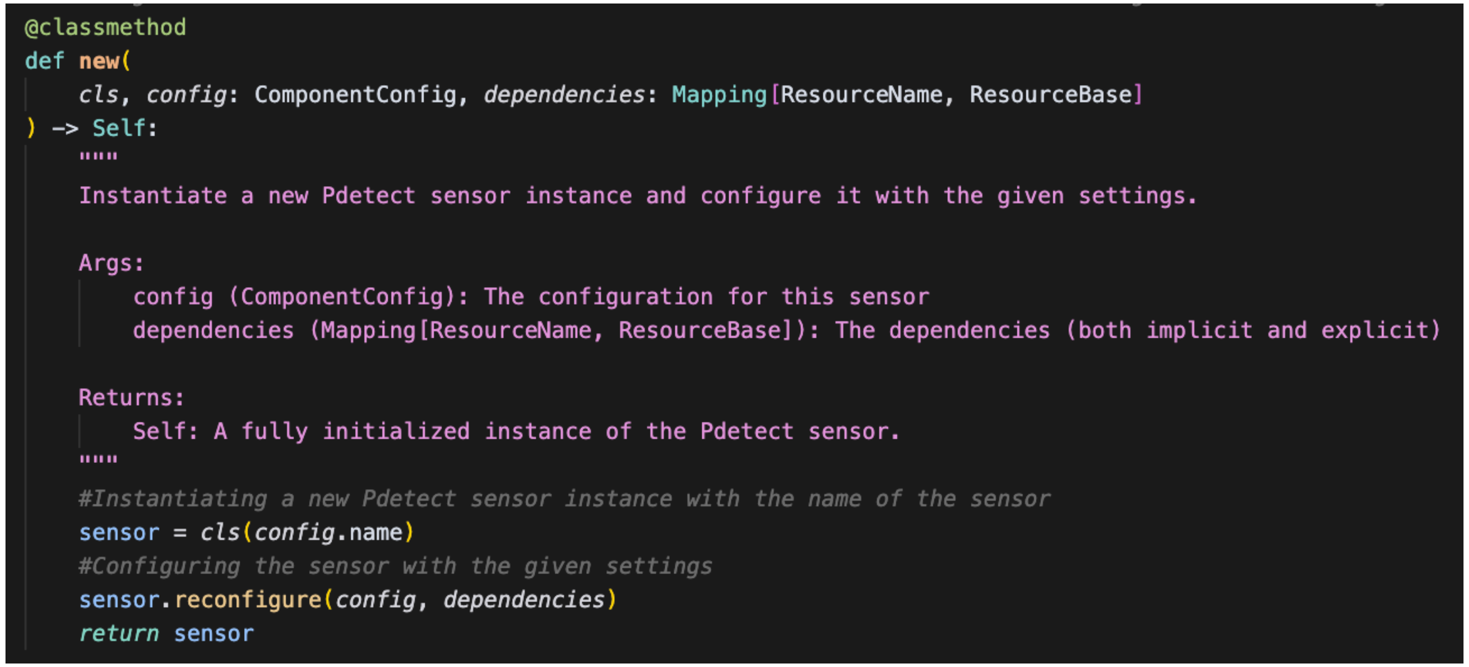
reconfigure()

do_command()

Task 3: Modular Registry: Implement the Component API

new():

- This method is responsible for creating and returning a new instance of the Pdetect sensor class.
    - It receives the config and dependencies from the Viam system.
    - It initializes the sensor (`sensor = cls(config.name)`) and then immediately calls `reconfigure()` to apply the configuration and wire up dependencies.
    - It returns the fully ready-to-use sensor instance.

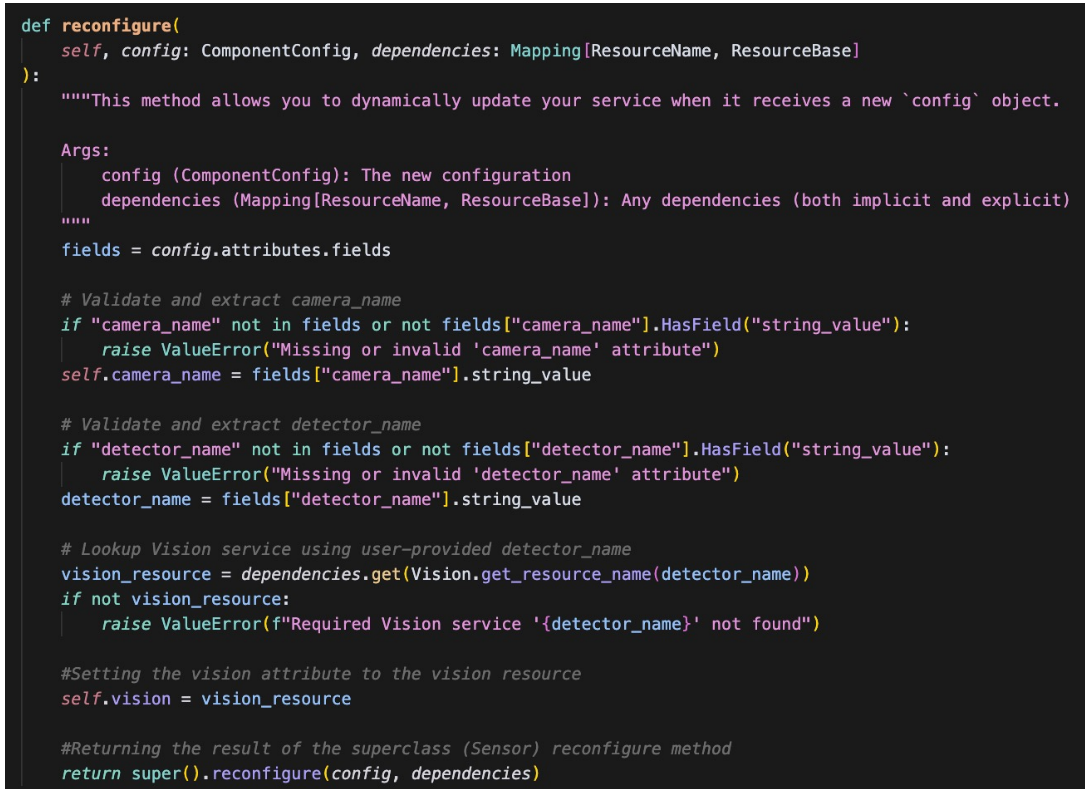


```python
@classmethod
def new(
    cls, config: ComponentConfig, dependencies: Mapping[ResourceName, ResourceBase]
) -> Self:
    """
    Instantiate a new Pdetect sensor instance and configure it with the given settings.

    Args:
        config (ComponentConfig): The configuration for this sensor
        dependencies (Mapping[ResourceName, ResourceBase]): The dependencies (both implicit and explicit)

    Returns:
        Self: A fully initialized instance of the Pdetect sensor.
    """
    #Instantiating a new Pdetect sensor instance with the name of the sensor
    sensor = cls(config.name)
    #Configuring the sensor with the given settings
    sensor.reconfigure(config, dependencies)
    return sensor
```

Task 3: Modular Registry: Implement the Component API

reconfigure():

- Triggered when the component's configuration is updated in the Viam app
  - Reads the latest `camera_name` and `detector_name` value from the incoming `config`
  - Retrieves the required dependency `detector_name` from the dependency map (throws error if not available)
  - Casts the required dependency as the correct type
  - Stores vision client and camera name internally as instance variables for use in detection logic in `do_command()`
  - Prepares the module for dynamic behavior based on configurable dependency names



```python
def reconfigure(
    self, config: ComponentConfig, dependencies: Mapping[ResourceName, ResourceBase]
):
    """This method allows you to dynamically update your service when it receives a new `config` object.

    Args:
        config (ComponentConfig): The new configuration
        dependencies (Mapping[ResourceName, ResourceBase]): Any dependencies (both implicit and explicit)
    """
    fields = config.attributes.fields

    # Validate and extract camera_name
    if "camera_name" not in fields or not fields["camera_name"].HasField("string_value"):
        raise ValueError("Missing or invalid 'camera_name' attribute")
    self.camera_name = fields["camera_name"].string_value

    # Validate and extract detector_name
    if "detector_name" not in fields or not fields["detector_name"].HasField("string_value"):
        raise ValueError("Missing or invalid 'detector_name' attribute")
    detector_name = fields["detector_name"].string_value

    # Lookup Vision service using user-provided detector_name
    vision_resource = dependencies.get(Vision.get_resource_name(detector_name))
    if not vision_resource:
        raise ValueError(f"Required Vision service '{detector_name}' not found")

    #Setting the vision attribute to the vision resource
    self.vision = vision_resource

    #Returning the result of the superclass (Sensor) reconfigure method
    return super().reconfigure(config, dependencies)
```

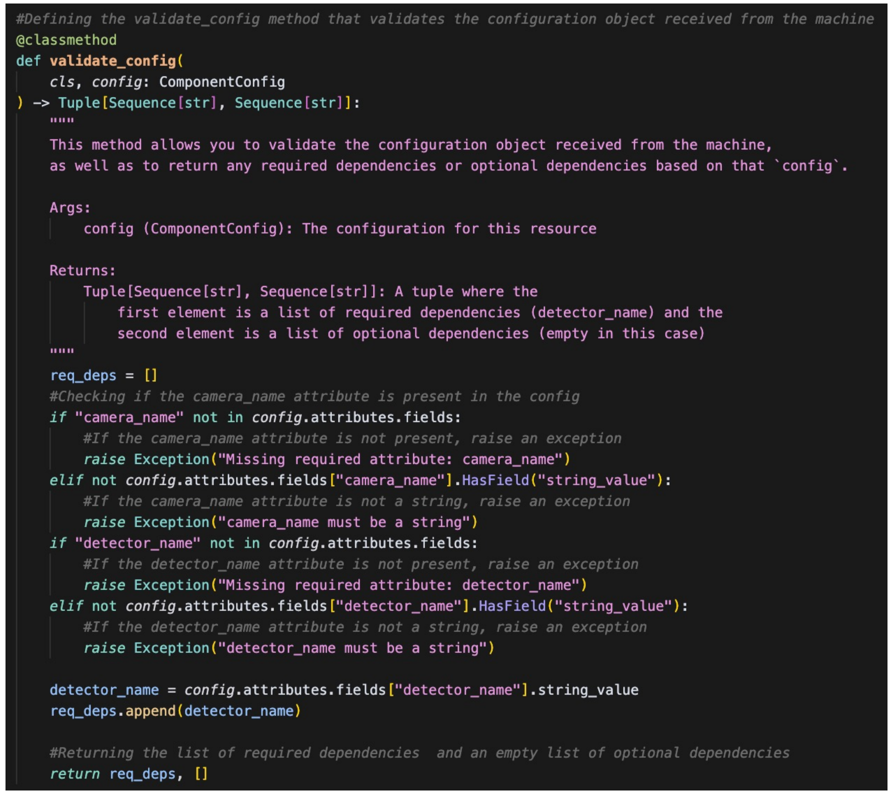
Task 3: Modular Registry: Implement the Component API

validate_config():

- Ensures that the user-supplied configuration is valid before the module is started.
    - Checks for required attributes: `camera_name` (required attribute) and `detector_name` (required attribute & dependency)
    - Raises errors if required fields are missing or improperly typed
- Declares required and optional dependencies for the module.
    - Dependencies returned: required: detector_name
    - This tells Viam to pass in the Vision service resource with this name at runtime.

Note:
`camera_name` is a required attribute because it's just a string used to route the vision query to the correct camera.

`detector_name` is a required dependency, since our module needs access to the actual Vision service object to perform detection.

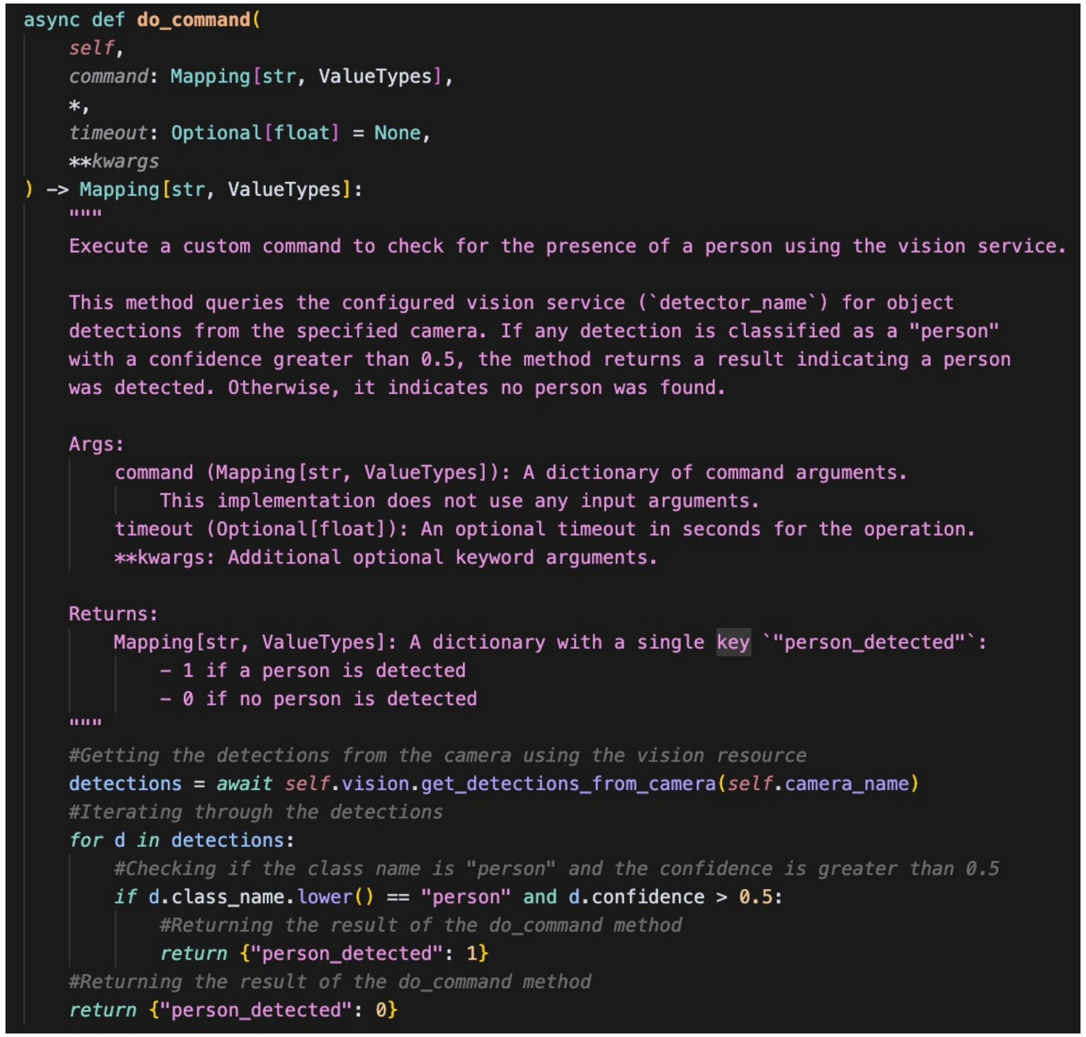


```python
#Defining the validate_config method that validates the configuration object received from the machine
@classmethod
def validate_config(
    cls, config: ComponentConfig
) -> Tuple[Sequence[str], Sequence[str]]:
    """
    This method allows you to validate the configuration object received from the machine,
    as well as to return any required dependencies or optional dependencies based on that `config`.

    Args:
        config (ComponentConfig): The configuration for this resource

    Returns:
        Tuple[Sequence[str], Sequence[str]]: A tuple where the
            first element is a list of required dependencies (detector_name) and the
            second element is a list of optional dependencies (empty in this case)
    """
    req_deps = []
    #Checking if the camera_name attribute is present in the config
    if "camera_name" not in config.attributes.fields:
        #If the camera_name attribute is not present, raise an exception
        raise Exception("Missing required attribute: camera_name")
    elif not config.attributes.fields["camera_name"].HasField("string_value"):
        #If the camera_name attribute is not a string, raise an exception
        raise Exception("camera_name must be a string")
    if "detector_name" not in config.attributes.fields:
        #If the detector_name attribute is not present, raise an exception
        raise Exception("Missing required attribute: detector_name")
    elif not config.attributes.fields["detector_name"].HasField("string_value"):
        #If the detector_name attribute is not a string, raise an exception
        raise Exception("detector_name must be a string")

    detector_name = config.attributes.fields["detector_name"].string_value
    req_deps.append(detector_name)

    #Returning the list of required dependencies  and an empty list of optional dependencies
    return req_deps, []
```

Task 3: Modular Registry: Implement the Component API

do_command():

- Allows the sensor to query the `myPeopleDetector` vision service and returns `person_detected: 1` if a person is seen with >0.5 confidence, otherwise 0.

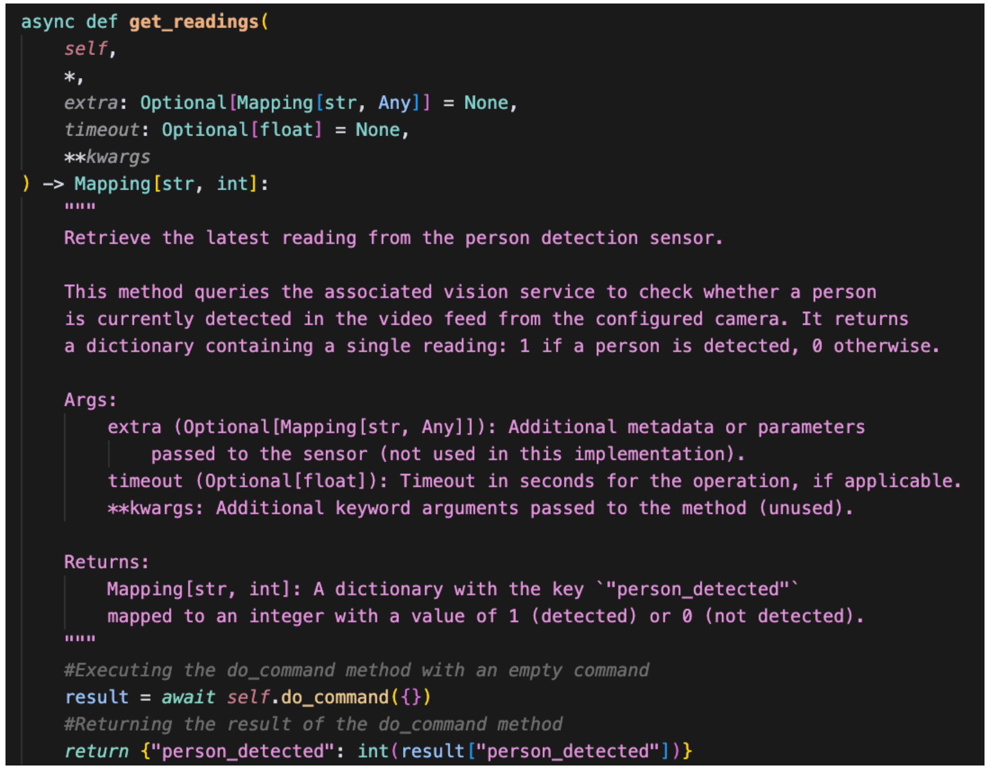


```python
async def do_command(
    self,
    command: Mapping[str, ValueTypes],
    *,
    timeout: Optional[float] = None,
    **kwargs
) -> Mapping[str, ValueTypes]:
    """
    Execute a custom command to check for the presence of a person using the vision service.

    This method queries the configured vision service (`detector_name`) for object
    detections from the specified camera. If any detection is classified as a "person"
    with a confidence greater than 0.5, the method returns a result indicating a person
    was detected. Otherwise, it indicates no person was found.

    Args:
        command (Mapping[str, ValueTypes]): A dictionary of command arguments.
            This implementation does not use any input arguments.
        timeout (Optional[float]): An optional timeout in seconds for the operation.
        **kwargs: Additional optional keyword arguments.

    Returns:
        Mapping[str, ValueTypes]: A dictionary with a single key `"person_detected"`:
            - 1 if a person is detected
            - 0 if no person is detected
    """
    #Getting the detections from the camera using the vision resource
    detections = await self.vision.get_detections_from_camera(self.camera_name)
    #Iterating through the detections
    for d in detections:
        #Checking if the class name is "person" and the confidence is greater than 0.5
        if d.class_name.lower() == "person" and d.confidence > 0.5:
            #Returning the result of the do_command method
            return {"person_detected": 1}
    #Returning the result of the do_command method
    return {"person_detected": 0}
```

Task 3: Modular Registry: Implement the Component API

- get_readings():
  - sensor interface that Viam calls to retrieve the latest data from the sensor.
  - Invokes the internal `do_command` method to perform the actual person detection logic.
- Wraps the result from `do_command` in a dictionary with a consistent sensor reading key:
  - Returns: `{"person_detected": 1}` if a person is detected, Returns: `{"person_detected": 0}` if no person is detected.

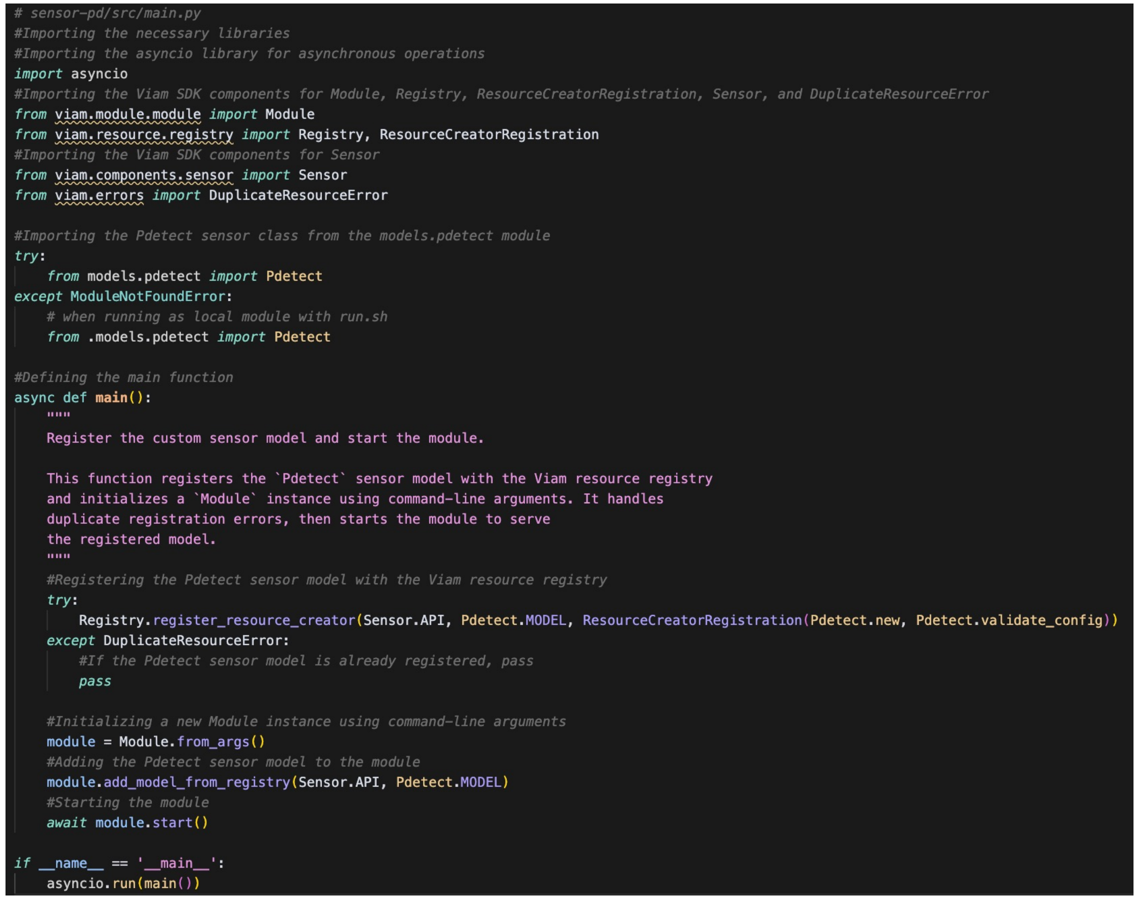
- This method ensures compatibility with the Viam Sensor API, allowing the module to be used like any other built-in Viam sensor.



```python
async def get_readings(
    self,
    *,
    extra: Optional[Mapping[str, Any]] = None,
    timeout: Optional[float] = None,
    **kwargs
) -> Mapping[str, int]:
    """
    Retrieve the latest reading from the person detection sensor.

    This method queries the associated vision service to check whether a person
    is currently detected in the video feed from the configured camera. It returns
    a dictionary containing a single reading: 1 if a person is detected, 0 otherwise.

    Args:
        extra (Optional[Mapping[str, Any]]): Additional metadata or parameters
            passed to the sensor (not used in this implementation).
        timeout (Optional[float]): Timeout in seconds for the operation, if applicable.
        **kwargs: Additional keyword arguments passed to the method (unused).

    Returns:
        Mapping[str, int]: A dictionary with the key `"person_detected"`
        mapped to an integer with a value of 1 (detected) or 0 (not detected).
    """
    #Executing the do_command method with an empty command
    result = await self.do_command({})
    #Returning the result of the do_command method
    return {"person_detected": int(result["person_detected"])}
```

Task 3: Modular Registry: Implement the Component API

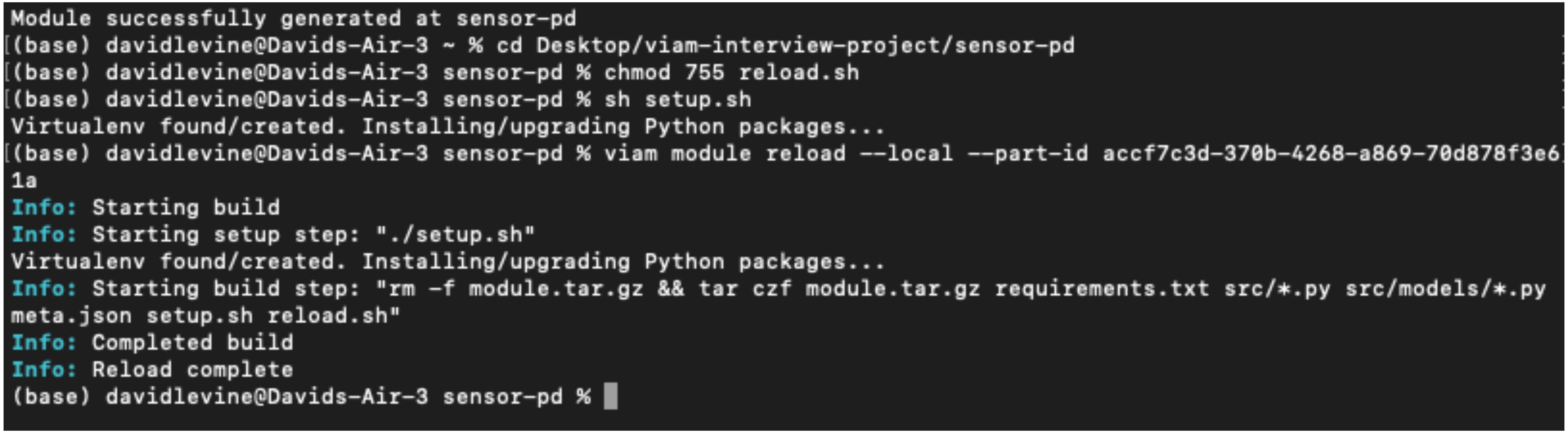
- main.py:
  - Uses Viam's registry to register the pdetect model under the Sensor API, linking the model name (`dl-org:sensor-pd:pdetect`) to my Pdetect implementation.
  - Includes error handling to avoid duplicate registration when hot reloading.
  - Attaches the registered model to the module so it can be discovered and used by Viam.
  - Initializes and launches the module to connect with the Viam cloud, enabling remote control and configuration.



```python
# sensor-pd/src/main.py
#Importing the necessary libraries
#Importing the asyncio library for asynchronous operations
import asyncio
#Importing the Viam SDK components for Module, Registry, ResourceCreatorRegistration, Sensor, and DuplicateResourceError
from viam.module.module import Module
from viam.resource.registry import Registry, ResourceCreatorRegistration
#Importing the Viam SDK components for Sensor
from viam.components.sensor import Sensor
from viam.errors import DuplicateResourceError

#Importing the Pdetect sensor class from the models.pdetect module
try:
    from models.pdetect import Pdetect
except ModuleNotFoundError:
    # when running as local module with run.sh
    from .models.pdetect import Pdetect

#Defining the main function
async def main():
    """
    Register the custom sensor model and start the module.

    This function registers the `Pdetect` sensor model with the Viam resource registry
    and initializes a `Module` instance using command-line arguments. It handles
    duplicate registration errors, then starts the module to serve
    the registered model.
    """
    #Registering the Pdetect sensor model with the Viam resource registry
    try:
        Registry.register_resource_creator(Sensor.API, Pdetect.MODEL, ResourceCreatorRegistration(Pdetect.new, Pdetect.validate_config))
    except DuplicateResourceError:
        #If the Pdetect sensor model is already registered, pass
        pass

    #Initializing a new Module instance using command-line arguments
    module = Module.from_args()
    #Adding the Pdetect sensor model to the module
    module.add_model_from_registry(Sensor.API, Pdetect.MODEL)
    #Starting the module
    await module.start()

if __name__ == '__main__':
    asyncio.run(main())
```

Task 3: Modular Registry: Local Testing

Goal: Test my custom module locally.

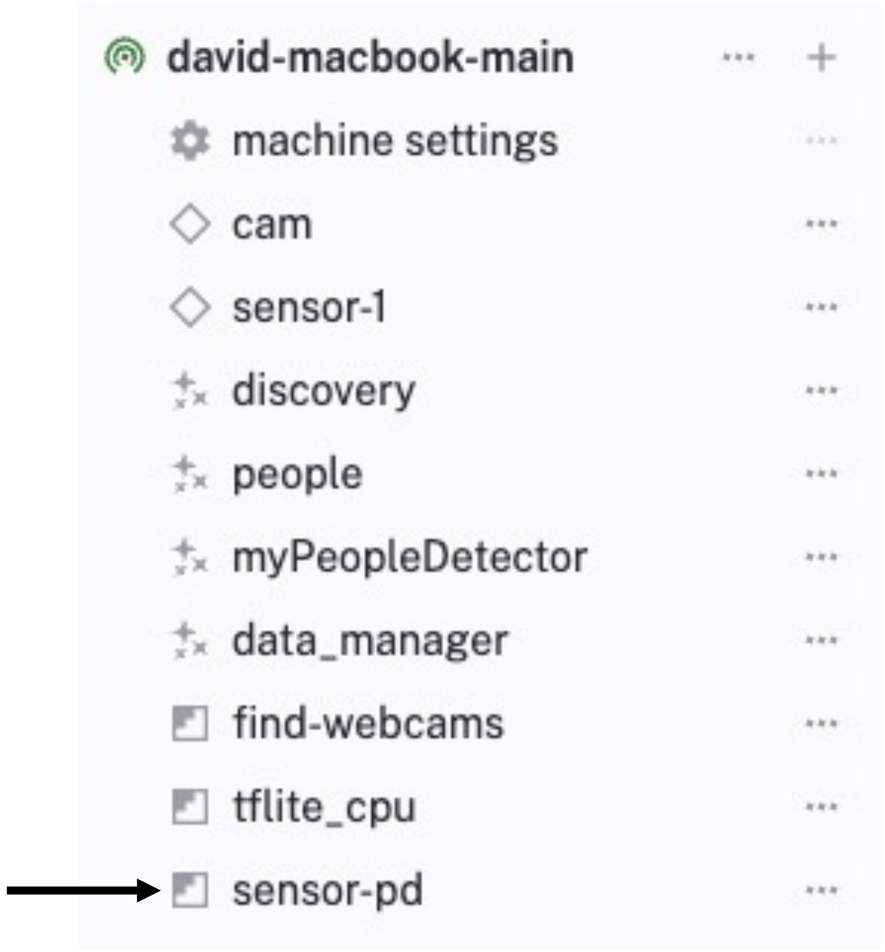


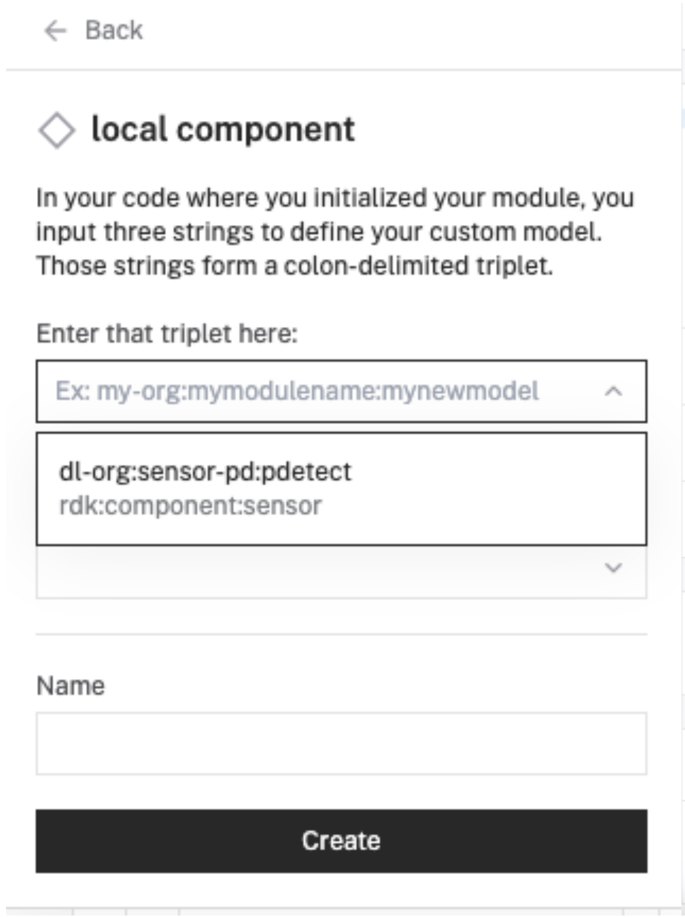
```
Module successfully generated at sensor-pd
[(base) davidlevine@Davids-Air-3 ~ % cd Desktop/viam-interview-project/sensor-pd
[(base) davidlevine@Davids-Air-3 sensor-pd % chmod 755 reload.sh
[(base) davidlevine@Davids-Air-3 sensor-pd % sh setup.sh
Virtualenv found/created. Installing/upgrading Python packages...
[(base) davidlevine@Davids-Air-3 sensor-pd % viam module reload --local --part-id accf7c3d-370b-4268-a869-70d878f3e6
1a
Info: Starting build
Info: Starting setup step: "./setup.sh"
Virtualenv found/created. Installing/upgrading Python packages...
Info: Starting build step: "rm -f module.tar.gz && tar czf module.tar.gz requirements.txt src/*.py src/models/*.py
meta.json setup.sh reload.sh"
Info: Completed build
Info: Reload complete
(base) davidlevine@Davids-Air-3 sensor-pd %
```

Steps to configure my local module on my machine

Task 3: Modular Registry: Adding & Testing the Sensor Component

Goal: Add my custom sensor component in the Viam UI.


david-macbook-main
machine settings
cam
sensor-1
discovery
people
myPeopleDetector
data_manager
find-webcams
tflite_cpu
sensor-pd


← Back

local component

In your code where you initialized your module, you input three strings to define your custom model. Those strings form a colon-delimited triplet.

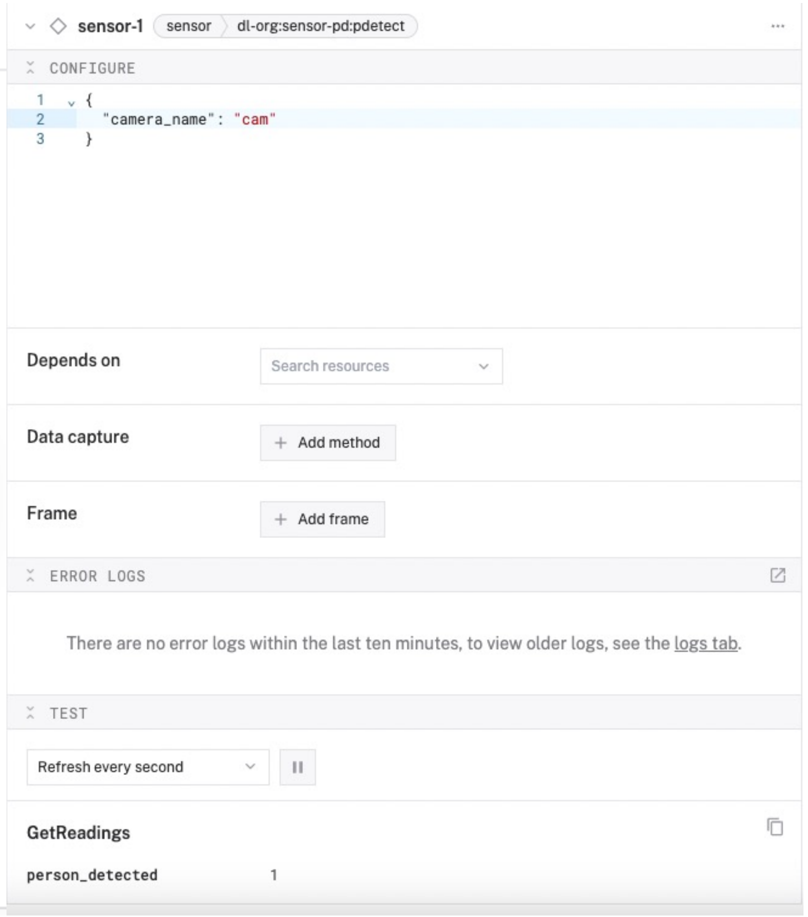
Enter that triplet here:

Ex: my-org:mymodulename:mynewmodel

dl-org:sensor-pd:pdetect
rdk:component:sensor

Name

Create


sensor-1 sensor dl-org:sensor-pd:pdetect

CONFIGURE

```
{
  "camera_name": "cam"
}
```

Depends on	Search resources
Data capture	+ Add method
Frame	+ Add frame

ERROR LOGS

There are no error logs within the last ten minutes, to view older logs, see the logs tab.

TEST

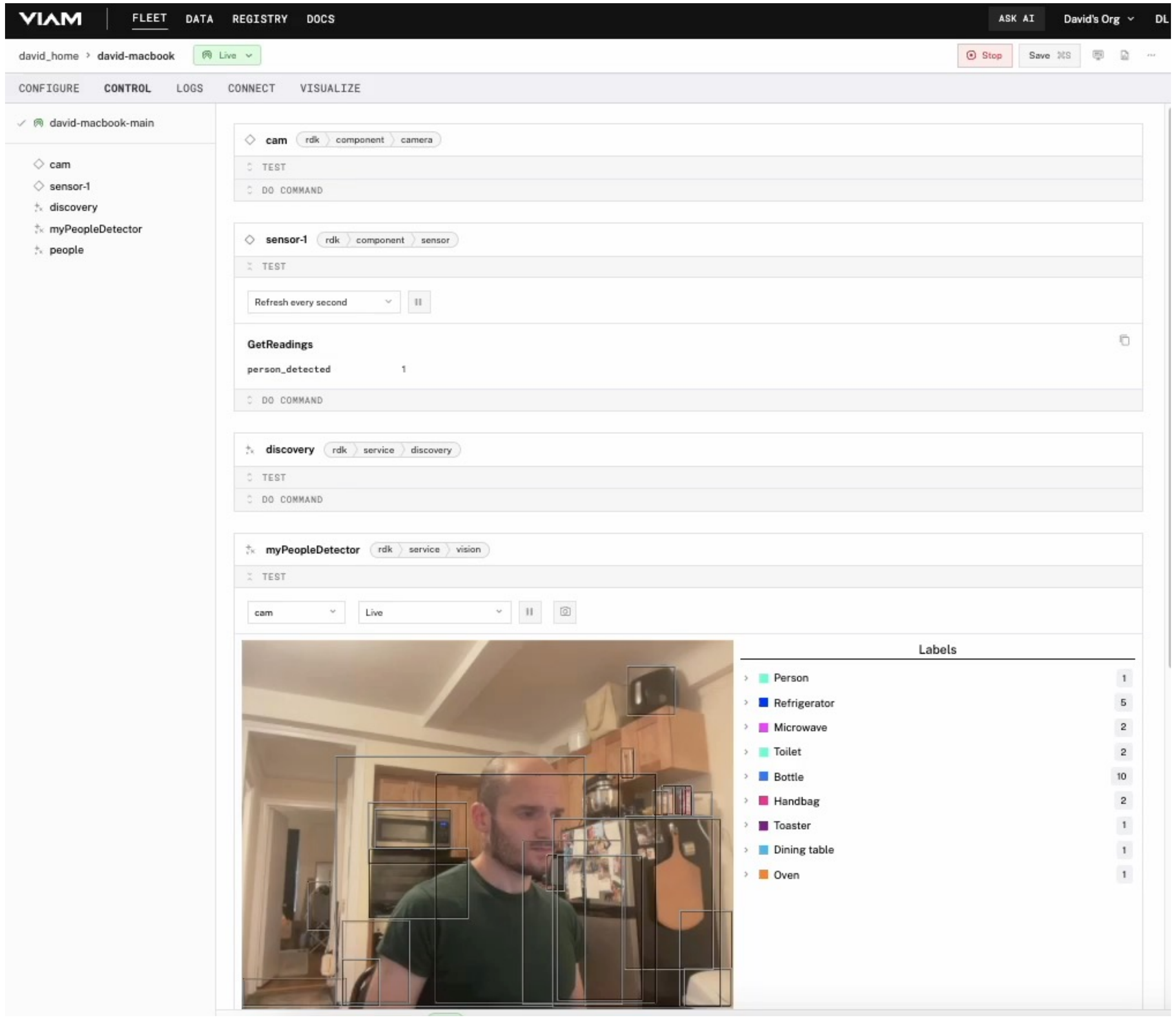
Refresh every second

GetReadings

```
person_detected    1
```

Task 3: Modular Registry: Adding & Testing the Sensor Component

Goal: Test my custom sensor component in the Viam UI.



```
GetReadings
person_detected          1
```

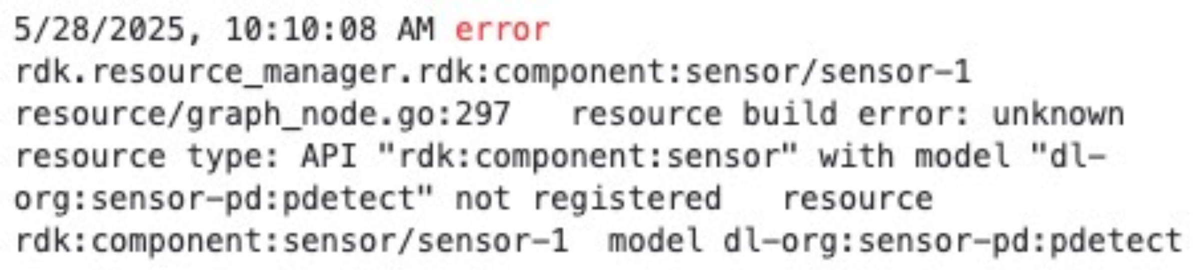
Labels

🟩	Person	1
🟦	Refrigerator	5
🟪	Microwave	2
🟩	Toilet	2
🟦	Bottle	10
🟥	Handbag	2
🟪	Toaster	1
🟦	Dining table	1
🟧	Oven	1

Task 3: Modular Registry: Error Handling

Goal: Find errors and fix bugs to complete Task 3.

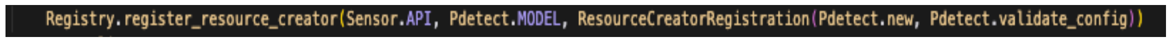
- Errors: `DuplicateResourceError: Cannot add resource with duplicate name "rdk:component:sensor/dl-org:sensor-pd:pdetect"` → caused registration error (below)
  - Cause:
    - The module attempted to register the same resource more than once during hot reloads.
  - Fix (**main.py**):
    - Wrapped the registration line in a try/except block to catch and suppress the DuplicateResourceError.



```
5/28/2025, 10:10:08 AM error
rdk.resource_manager.rdk:component:sensor/sensor-1
resource/graph_node.go:297   resource build error: unknown
resource type: API "rdk:component:sensor" with model "dl-
org:sensor-pd:pdetect" not registered   resource
rdk:component:sensor/sensor-1  model dl-org:sensor-pd:pdetect
```

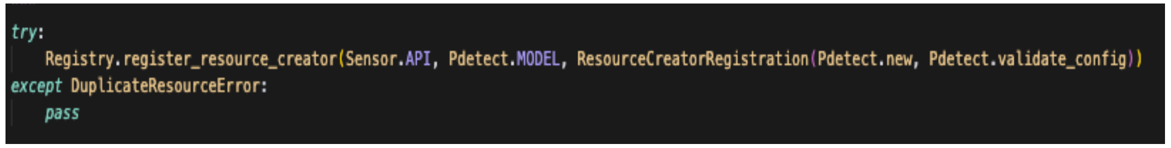
Registration error caused by duplicate registration

Before fix:



```
    Registry.register_resource_creator(Sensor.API, Pdetect.MODEL, ResourceCreatorRegistration(Pdetect.new, Pdetect.validate_config))
```

After fix:



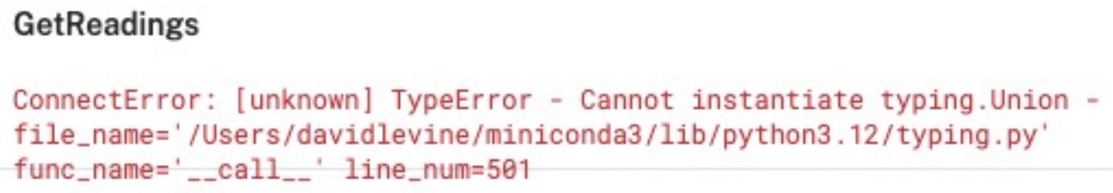
```
try:
    Registry.register_resource_creator(Sensor.API, Pdetect.MODEL, ResourceCreatorRegistration(Pdetect.new, Pdetect.validate_config))
except DuplicateResourceError:
    pass
```

try/except block to catch and suppress the DuplicateResourceError

Task 3: Modular Registry: Error Handling

Goal: Find errors and fix bugs to complete Task 3.

- Error: `TypeError - Cannot instantiate typing.Union`
  - Cause:
    - Attempted to wrap a basic type (`int`) inside `SensorReading`, which isn't necessary and led to incompatible typing. (SensorReading is a type alias)
  - Fix (**pdetect.py → get_readings()** method):
    - Removed the use of `SensorReading()` and returned a plain `int` instead.



GetReadings

```
ConnectError: [unknown] TypeError - Cannot instantiate typing.Union -
file_name='/Users/davidlevine/miniconda3/lib/python3.12/typing.py'
func_name='__call__' line_num=501
```

Type Error found in sensor_1 log

Before fix:

```
return {"person_detected":
SensorReading(int(result["person_detected"]))}
```

After fix:

```
return {"person_detected":
int(result["person_detected"])}
```

Removed the use of `SensorReading()`

Next Steps: Upload Module to Viam Registry

1. Set your module to public or private visibility.
When running `viam module generate`, choose whether others can access the module.

2. Update meta.json.
Ensure the entrypoint, build, path, and supported arch values are set correctly.
This file tells Viam how to build, package, and run the module.

3. Package your module.
Run the build command (or use build.sh) to create a module.tar.gz file.

4. Login to Viam.

5. Register your module.

6. Upload your module.
Push the module to the registry.

7. Use the module in the cloud.

Opportunities for Improvement

- Customizable Configuration via Viam UI
  - Support configurable confidence thresholds, detection classes, and different camera inputs.

- Capture & Store Data with Viam
  - Extend to support other ML models beyond people detection in the Vision service.
  - Integrate with Viam's Data Capture UI to view and download results.

- Build a UI for Sensor Control
  - Develop a lightweight frontend to:
    - Display live detection results from `get_readings`.
    - Visualize logs and sensor state.

- Robust Error Handling
  - Add better exception management, validation, and user-visible error messages.

- Developer Experience & Docs
  - Publish module to Viam Registry.
  - Include links to my GitHub, Docs, and example `meta.json` in the Registry.