Cloud Clinic 2

Data Publication

27-FEB-2025

Rob Fatland (rob5@uw.edu), Naomi Alterman

“Shoebox to Science Gateway: Data publication and API access”

```
$ git clone https://github.com/robfatland/oceanclient
```

CloudBank "Cloud Clinic" series

- Cloud Clinics: build-path feasibility
  - Data science environments on public cloud platforms

- Clinic 1: Massive cost savings from preemptible instances

- Clinic 2: Science Gateway: data publication and access
  - Simple: Periodic table of elements
  - Complex: Ocean sensor data

- Jargon: NoSQL, Serverless, API, VSCode

- ~~recipe~~ 'Knowing enough to build with confidence' sub-text

```
$ git clone https://github.com/robfatland/oceanclient
```

Cloud Clinic 2 Abstract

Organizations such as Science Gateways and the eScience Institute idealistically promote open science through data sharing; and you may wish you had the skills to build something that puts you firmly in that camp. Go open science! But there is a catch: Building something that works is much easier than building something that works that is secure. And then there is the inevitable catastrophe once you have it up and running: You have a new idea and you wish to expand on what your system's baseline design was intended to do. No fear: This clinic will give you the basic one-two-three punch to build a data server with a built-in API, make it secure enough (assuming you are not working with personalized human data), and expand it in a new direction after it is up and running. We will use as a working example the supposition that you have invented the periodic table of elements and that you subsequently discovered crystal field theory. We address the pressing question: Can a cloud-hosted NoSQL chemistry data system be ACIDic? **Atomic Consistent Isolated Durable**

```
$ git clone https://github.com/robfatland/oceanclient
```

Who Is Giving This Talk

The narrator is not a computer scientist

The narrator does have experience with shoeboxes.

The narrator subscribes to the open science philosophy

```
$ git clone https://github.com/robfatland/oceanclient
```

Rob's First Law (R_1)

Data is never acquired in the manner in which it is used.

The Shoebox Problem

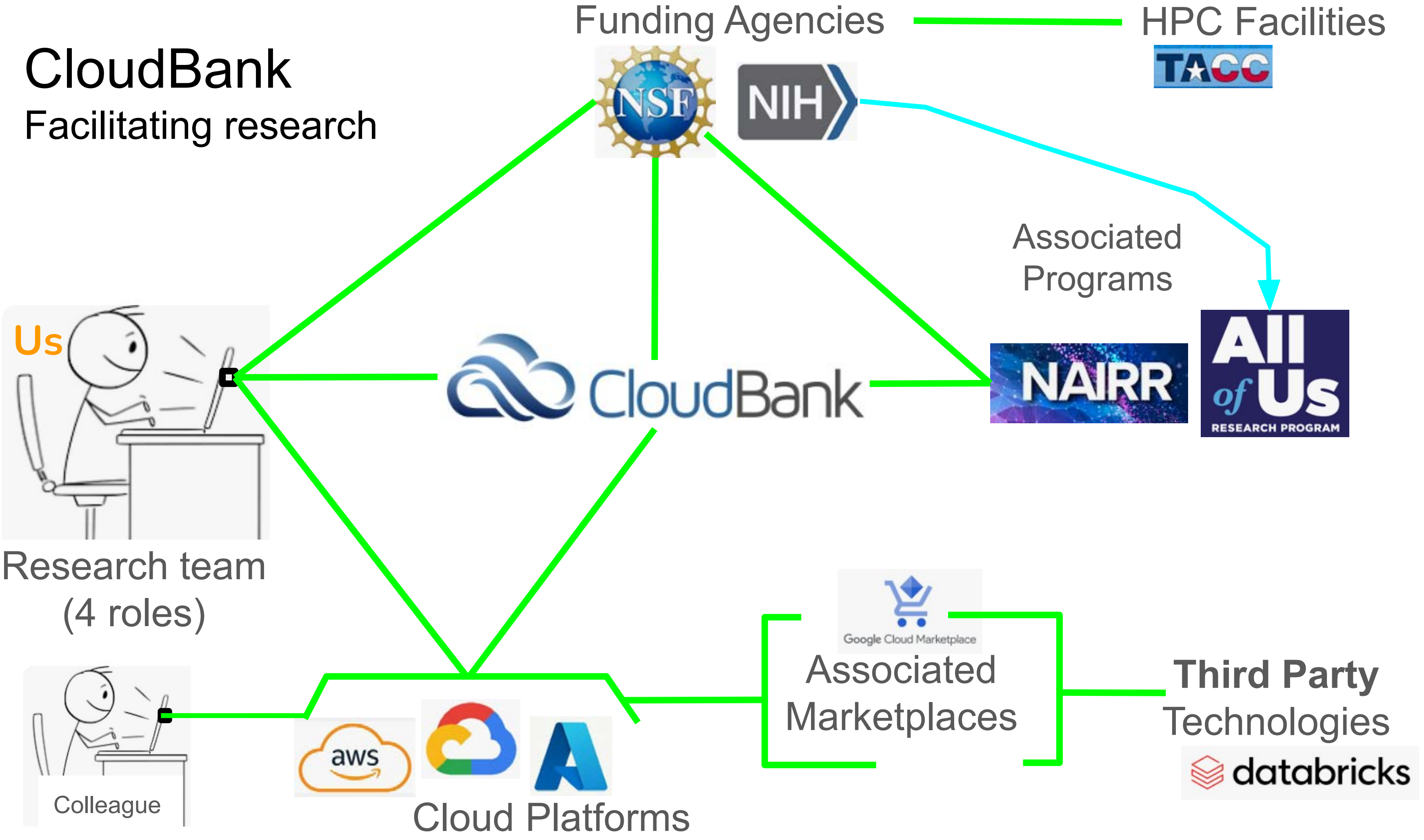
Hey look what I found under the desk! A shoebox of data tapes! Gosh it would be cool to publish this data on the web for open use… but how?

Digression: Cloud platforms for data science

- Research roles
  - Principle Investigator
  - Administrator$_{\$}$
  - Builders (perception: lot of work!)
  - Users (including external/unknown Users)
- Case study
  - Ocean observatory: One-sample-per-second data from sensors
  - Scientist has a “2 lines of code” view of this data
- Demystify data publication and access

…and now a moment of organization structure…

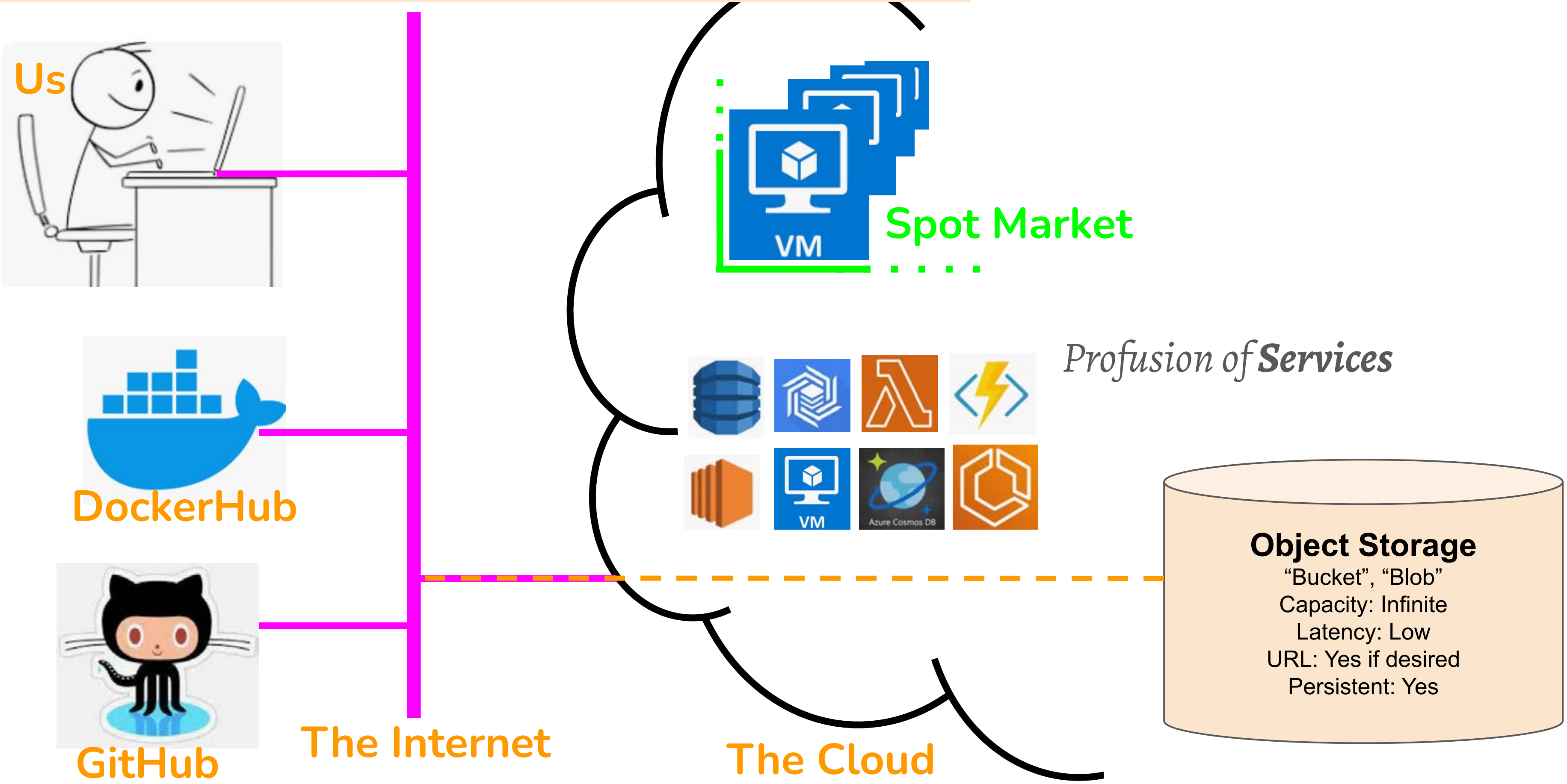
CloudBank
Facilitating research



CloudBank Support Framework

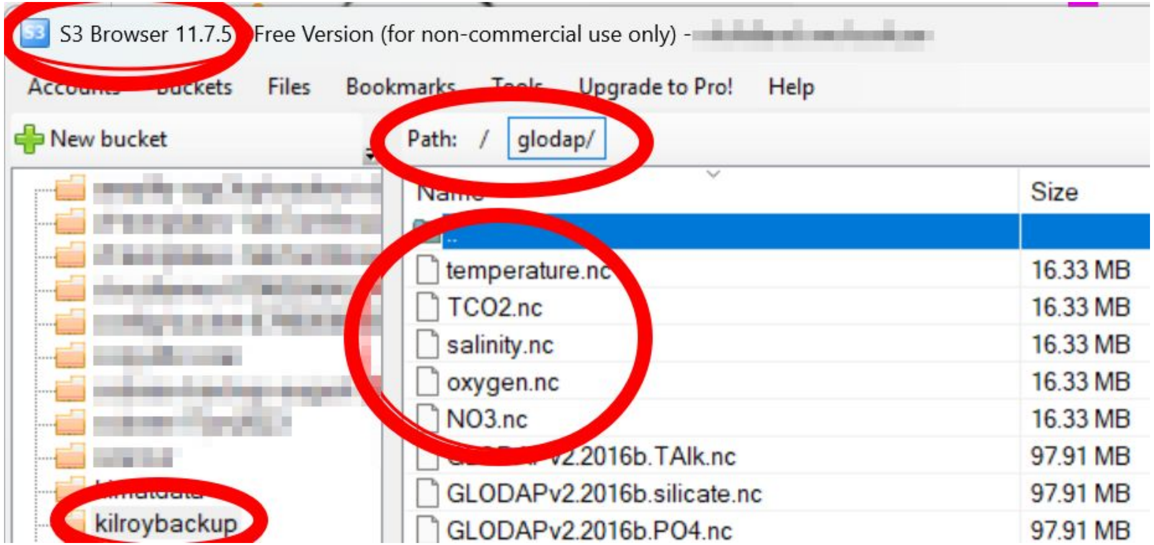
- Portal https://cloudbank.org
- Learning https://cloudbank-project.github.io/cb-resources/
- Community https://community.cloudbank.org/
- Studies… example SkyPilot: https://github.com/oorjitchowdhary/cifar-on-spot-vm

The Cloud-for-Research Ecosystem



Returning to today’s topic:

Public object storage “Wheel your data out to the curb”



Done!

Approaches to data publication: Access implications

Where the data are published	**Advantages**	**Disadvantages**
A shoebox or USB drive	Low cost (USPS media rate), low effort	Does not scale, does not address R_1
Google Drive, OneDrive, DropBox etcetera	Pretty easy on the effort scale, access is intuitive and can be managed by Share	Limited volume, hard to cite/find, does not scale, does not address R_1
Cloud object storage: S3 bucket, etcetera	Infinite volume, pretty cheap, better security with some added cloud machinery	No flexibility, low baseline security. The burden is on the Downloader to make sense of the data.
Cloud (No)SQL Database + Virtual Machine	Flexible, scales, addresses R_1, secure, good example of open science, probably fun	Maintenance of operating cloud virtual machines at scale (patches etc); can be more costly than (5); track and cover cost of operation
Cloud NoSQL Database + serverless API	Flexible, scales, addresses R_1 , secure, leadership by example in open science, opens doors to collaboration, definitely fun	Time investment to learn, build and maintain the technology; must track and cover cost of operation

Data Publication and Access

Options beyond wheeling data out into the street for anyone to download...

Here we persevere to ‘serverless’ with two { simple, complex} examples

Simple: Publish “sparse/wide” periodic table: id, cols. Query via API (browser):

https://pythonbytes.azurewebsites.net/api/lookup?name=Sodium

Complex: Publish data from a UW-based ocean observatory. Access by API; but now from Python: Use a published Client and just 2 lines of code

https://oceansensors.azurewebsites.net/api/sensors?start=2022-01-02%2010:00:00&stop=2022-01-02%2010:00:02

Where to begin: Naomi Alterman’s MSE544 periodic table walkthrough

Where to begin: Naomi's MSE544 Walkthrough

https://cloudbank-project.github.io/az-serverless-tutorial/

Serverless Azure Tutorials

This website hosts a series of tutorials explaining how to
modules are intended to be followed in the order presen

Modules

1. VMs and Workstations
2. NoSQL Databases
3. Serverless Functions and APIs

Simple

https://pythonbytes.azurewebsites.net/api/lookup?name=Sodium

Pretty-print ☑


```
[
  {
    "AtomicNumber": 11,
    "Element": "Sodium",
    "Symbol": "Na",
    "AtomicMass": 22.99,
    "NumberOfNeutrons": 12,
    "NumberOfProtons": 11,
    "NumberOfElectrons": 11,
    "Period": 3,
    "Group": "1",
    "Phase": "Solid",
    "Radioactive": false,
    "Natural": true,
    "Metal": true,
    "Nonmetal": false,
    "Metalloid": false,
    "Type": "Alkali Metal",
    "AtomicRadius": 227,
    "Electronegativity": 0.93,
    "ionizationEnergy": 5.1391,
    "Density": 0.97,
    "MeltingPoint": 370.95,
    "BoilingPoint": 1156,
    "stableIsotopes": 1,
    "Discoverer": "Sir Humphrey Davy",
    "Year": 1807,
    "SpecificHeat": 1.228,
    "NumberOfShells": 3,
    "NumberOfValence": 1,
    "id": "Sodium"
  }
```

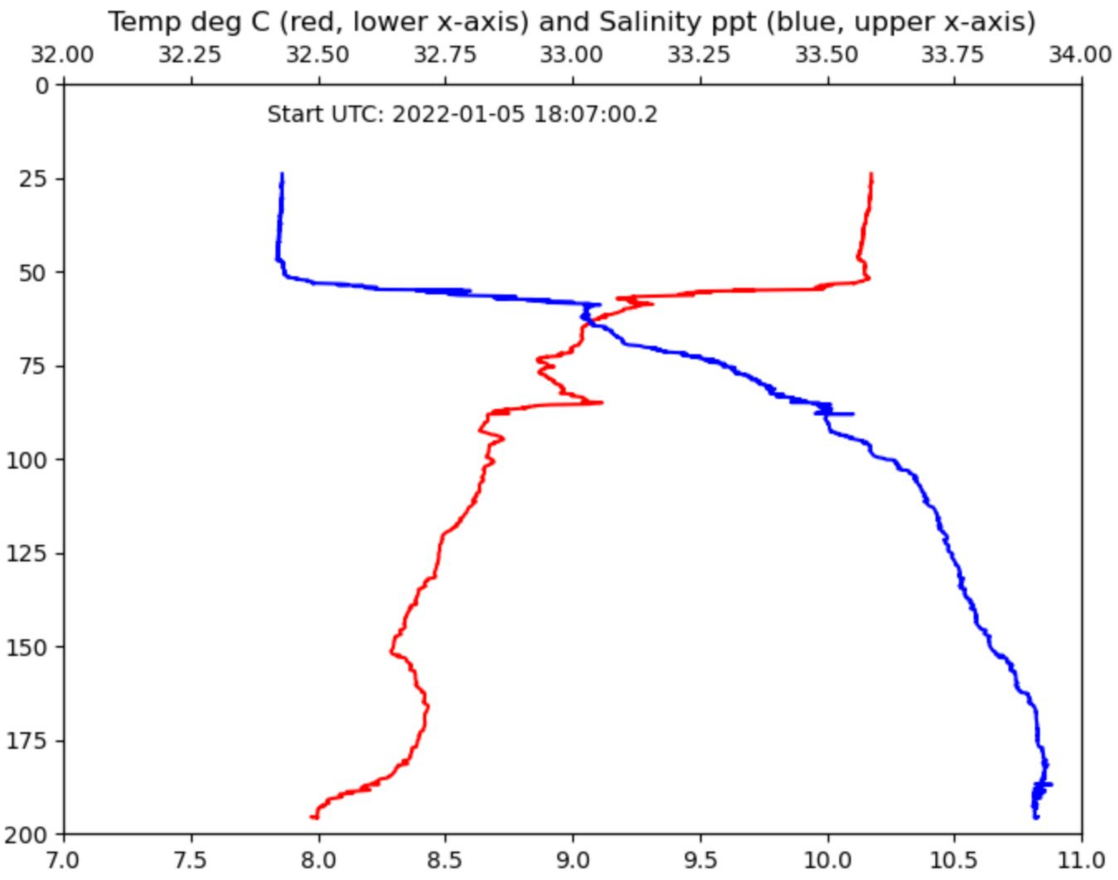
Complex: The road to ‘*two lines of code*’

- Stage data in tabular / CSV form
- Configure and pre-load a NoSQL database
- Write and test a data access API: Publish as a serverless function
- Write and test a Client that uses this API
- Publish the Client: GitHub repo or as a Python library
- Colleague: `$ git clone https://github.com/my-org/oceanclient`

and voila…

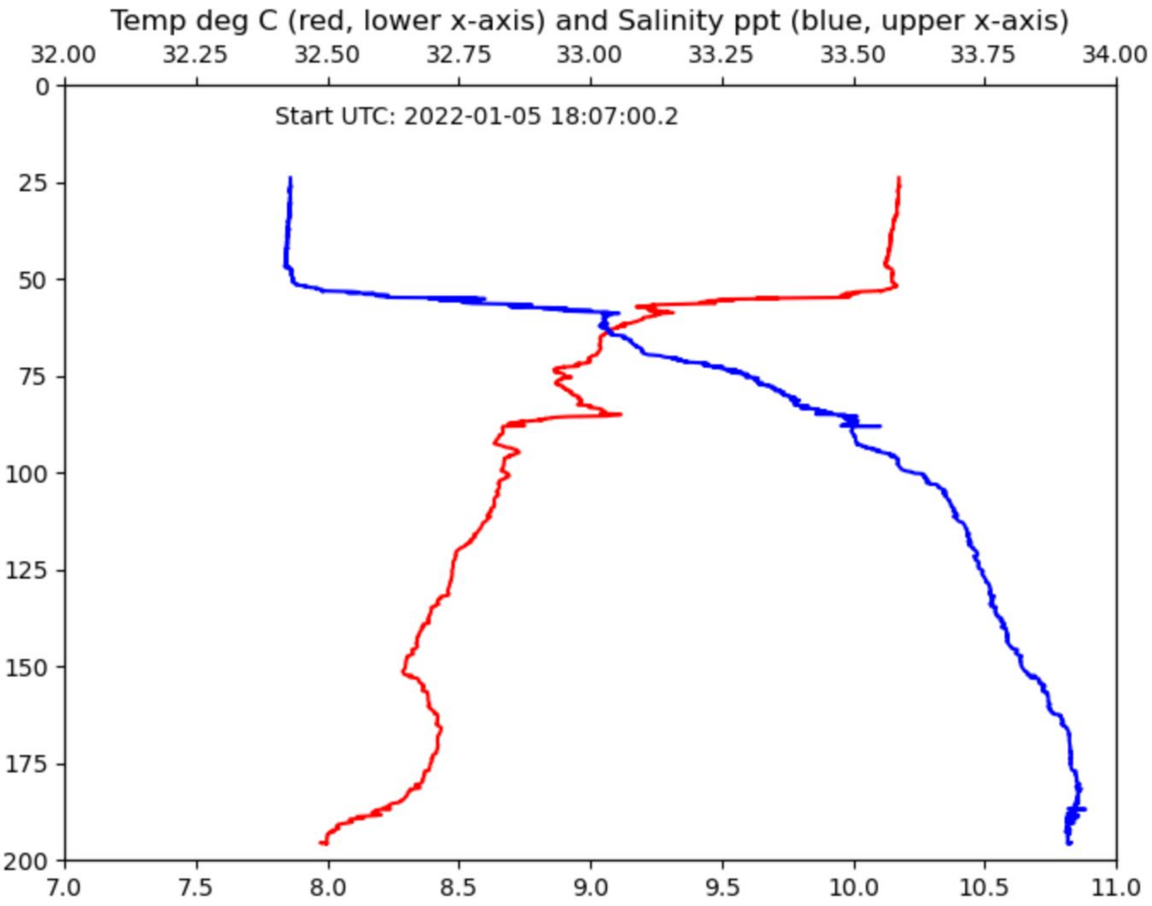
```
import oceanclient as oc
dfT, dfS = oc.Chart('2022-01-05', 9)
```

Chart



Temp deg C (red, lower x-axis) and Salinity ppt (blue, upper x-axis)
32.00
32.25
32.50
32.75
33.00
33.25
33.50
33.75
34.00
Start UTC: 2022-01-05 18:07:00.2
0
25
50
75
100
125
150
175
200
7.0
7.5
8.0
8.5
9.0
9.5
10.0
10.5
11.0

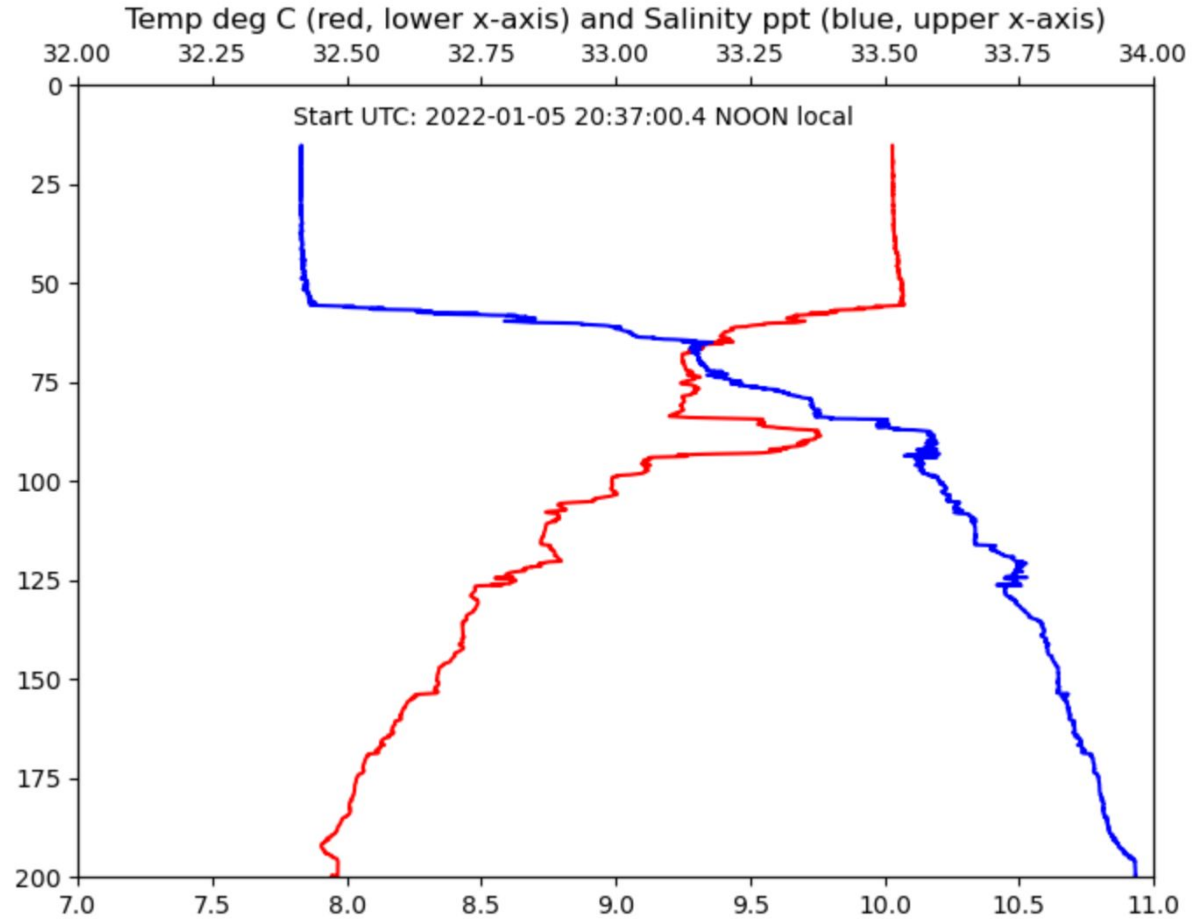
Chart



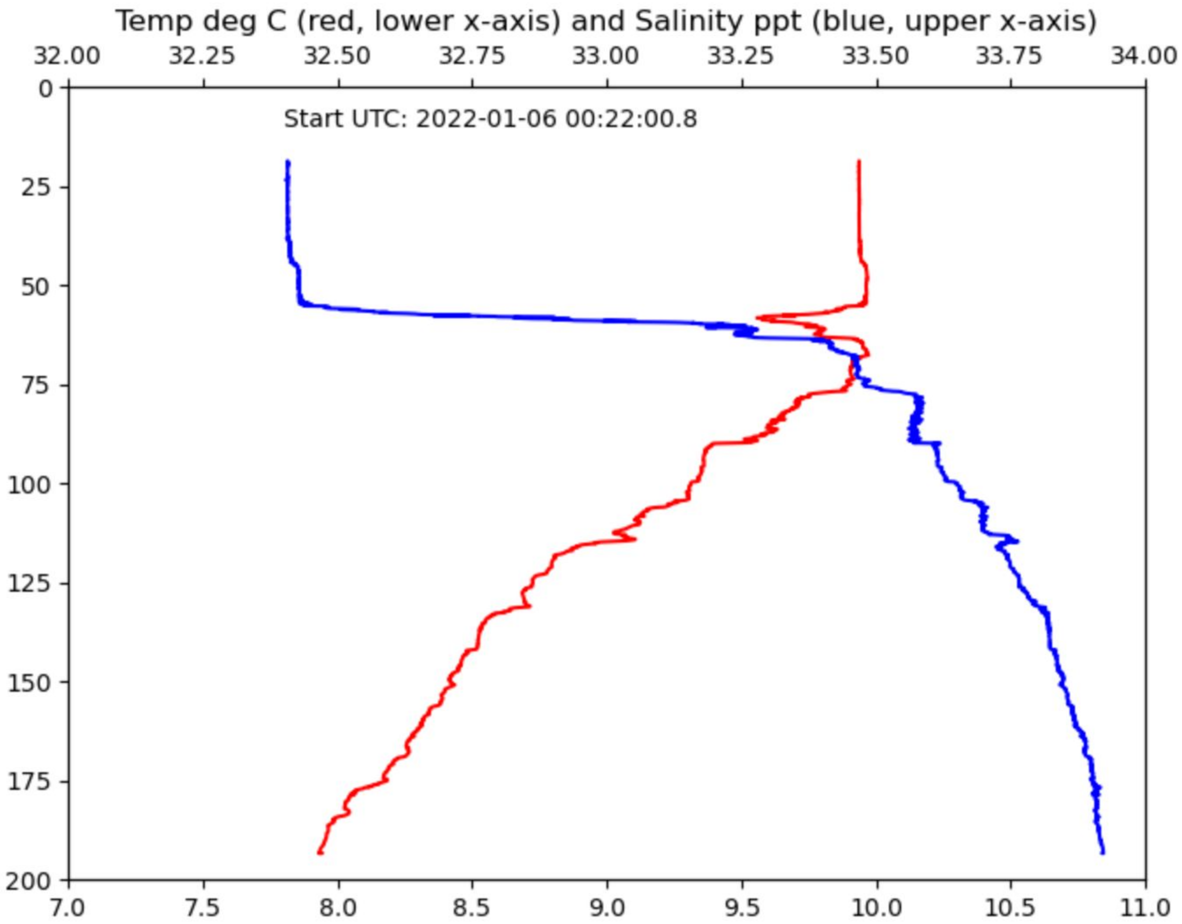
Temp deg C (red, lower x-axis) and Salinity ppt (blue, upper x-axis)
32.00
32.25
32.50
32.75
33.00
33.25
33.50
33.75
34.00
Start UTC: 2022-01-05 18:07:00.2
0
25
50
75
100
125
150
175
200
7.0
7.5
8.0
8.5
9.0
9.5
10.0
10.5
11.0

Chart

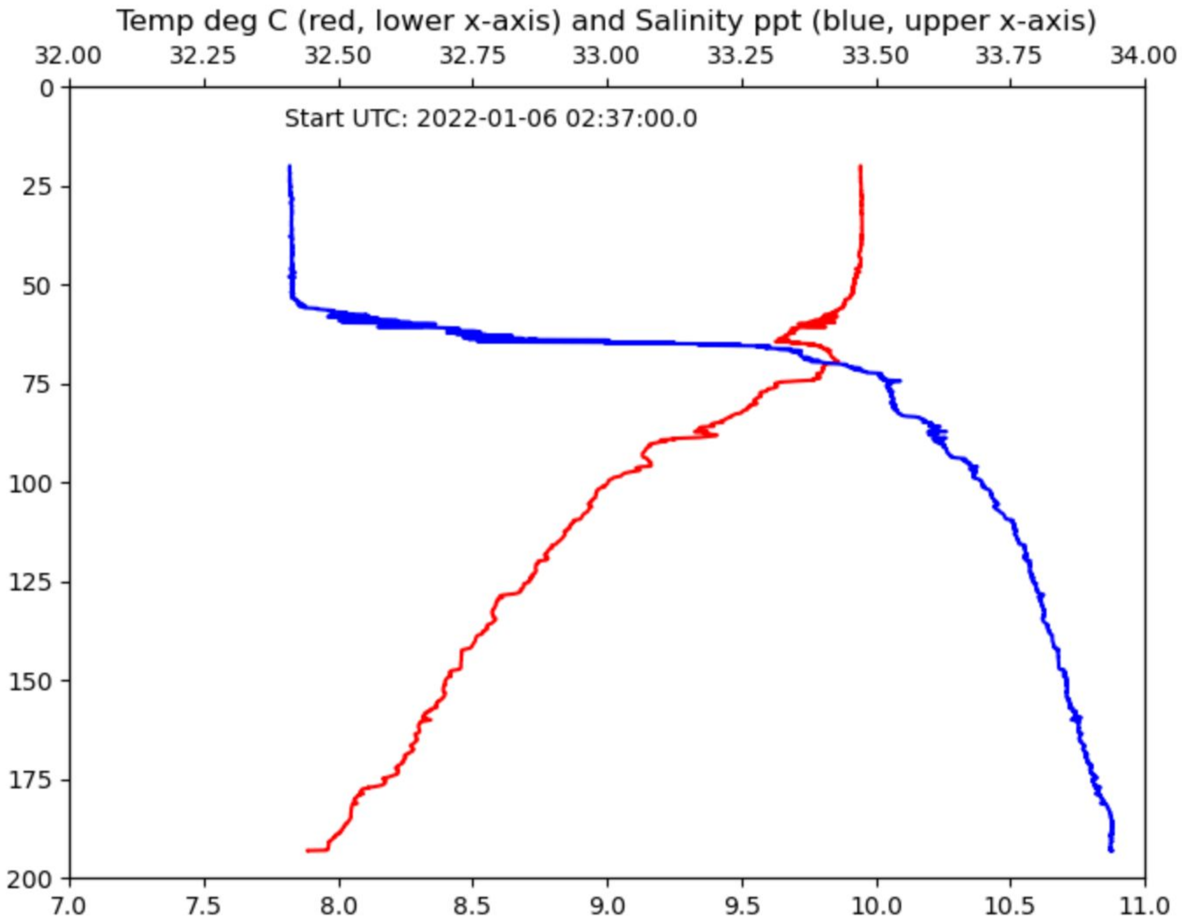
```
prep time 6.17 seconds; data vector length: 4380
```



Chart

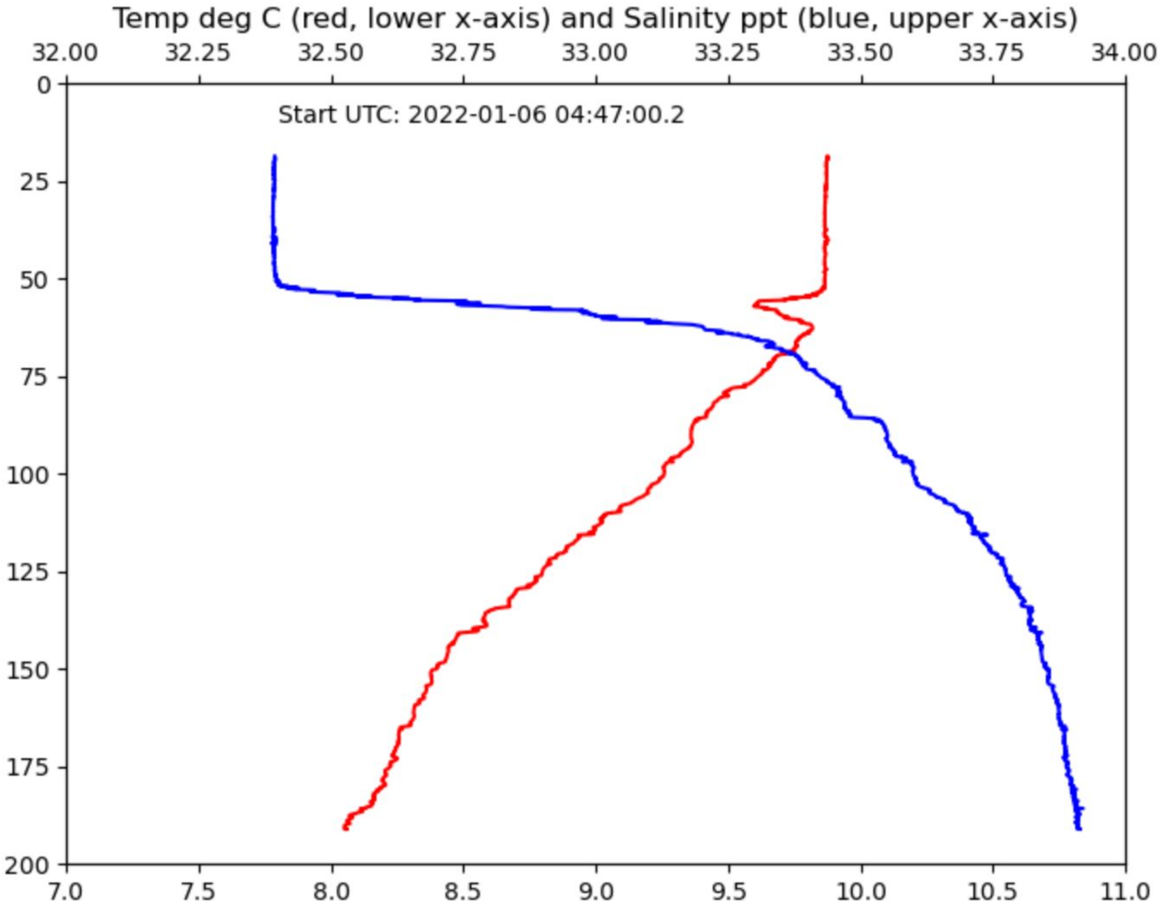

Temp deg C (red, lower x-axis) and Salinity ppt (blue, upper x-axis)
32.00
32.25
32.50
32.75
33.00
33.25
33.50
33.75
34.00
Start UTC: 2022-01-06 00:22:00.8
0
25
50
75
100
125
150
175
200
7.0
7.5
8.0
8.5
9.0
9.5
10.0
10.5
11.0

Chart



Temp deg C (red, lower x-axis) and Salinity ppt (blue, upper x-axis)
32.00
32.25
32.50
32.75
33.00
33.25
33.50
33.75
34.00
Start UTC: 2022-01-06 02:37:00.0
0
25
50
75
100
125
150
175
200
7.0
7.5
8.0
8.5
9.0
9.5
10.0
10.5
11.0

Chart



Temp deg C (red, lower x-axis) and Salinity ppt (blue, upper x-axis)
32.00
32.25
32.50
32.75
33.00
33.25
33.50
33.75
34.00
Start UTC: 2022-01-06 04:47:00.2
0
25
50
75
100
125
150
175
200
7.0
7.5
8.0
8.5
9.0
9.5
10.0
10.5
11.0

Data

```
[4]: dfS
```

[4]:

	Timestamp	depth	salinity
0	2022-01-05 20:37:00.482559488	199.660778	33.967098
1	2022-01-05 20:37:01.482462720	199.662944	33.967234
2	2022-01-05 20:37:02.482989568	199.664009	33.967048
3	2022-01-05 20:37:03.482579456	199.659779	33.966984
4	2022-01-05 20:37:04.482899456	199.655482	33.966795
...	...	...	...
4375	2022-01-05 21:49:55.597626880	15.886139	32.415099

Time to build: Periodic Table example

- Cloud subscription, log in to the portal, navigate: 2 hours + admin time
- Start a cloud VM, log in, run some commands: 2 hours
- Start a NoSQL database, install data (periodic table): 2 hours
- Create an Azure Function App: 2 hours
- Wire it all up: 2 hours

Total with overhead, background reading, non-recipe approach: 2 days

After this time investment one has a *very good grasp of the process*

New to cloud infrastructure: More background learning

Experienced with cloud: One day

Build a sophisticated custom data system: months

Resources

MSE544: https://cloudbank-project.github.io/az-serverless-tutorial/

Internet-2: C.L.A.S.S. Cloud Learning And Skills Sessions

The Carpentries: Basics of `git`, `bash`, `Python`

`nexus`: annotation of Simple and Complex cases

your browser search window

https://github.com/robfatland/oceanclient

What is nexus?

'Repetition legitimizes… repetition legitimizes…' -Adam Nealy

nexus is a GitHub repo

…the narrator's ***process notes***…

how to, pointers, annotations, gotchas

(the details I forget after 2 days)

Verify the version of Ubuntu using `lsb_release -a`.

This block installs the `miniconda` package.

```
cd ~
which python3
git clone https://github.com/robfatland/ant
mkdir -p ~/miniconda3
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O ~/miniconda3/minoconda.sh
bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
rm ~/miniconda3/miniconda.sh
```

To ensure access to `miniconda` from the command line, place the following line at the very end of `~/.bashrc`:

```
export PATH=~/miniconda3/bin:$PATH
```

https://robfatland.github.io/nexus/data/api

On R_1 and scale

Two aspects of *scale*

- Volume: accommodate addition of more data
- Voracity: accommodate a community's growing data demand

R_1 : A deep topic, core = data cleaning, formatting, synthesis

'How much effort is needed to get data into a shareably useful format?'

Approaches to data publication: Access implications

Where the data are published	**Advantages**	**Disadvantages**
A shoebox or USB drive	Low cost (USPS media rate), low effort	Does not scale, does not address R_1
Google Drive, OneDrive, DropBox etcetera	Pretty easy on the effort scale, access is intuitive and can be managed by Share	Limited volume, hard to cite/find, does not scale, does not address R_1
Cloud object storage: S3 bucket, etcetera	Infinite volume, pretty cheap, better security with some added cloud machinery	No flexibility, low baseline security. The burden is on the Downloader to make sense of the data.
Cloud (No)SQL Database + Virtual Machine	Flexible, scales, addresses R_1, secure, good example of open science, probably fun	Additional maintenance overhead and cost operating cloud virtual machines at scale: Installing patches etcetera; must track and cover cost.
Cloud NoSQL Database + serverless API	Flexible, scales, addresses R_1 , secure, leadership by example in open science, opens doors to collaboration, definitely fun	Time investment to learn, build and maintain the technology; must track and cover cost

Simple Goal: Publish the periodic table of elements

…and provide an API; test from a browser tab or code…

This website hosts a series of tutorials explaining how to use Microsoft's Visual Studio Code editor and Azure cloud to create a low-cost serverless web API. These tutorial modules are intended to be followed in the order presented below.

Modules

1. VMs and Workstations

2. NoSQL Databases

3. Serverless Functions and APIs

Complex goal: Interrelated data and metadata

We have profile metadata and observational data from two sensors

Next: Review the basic build / collaborate structure

Then: A demo

Finally: Some details we hope are of interest, Q&A

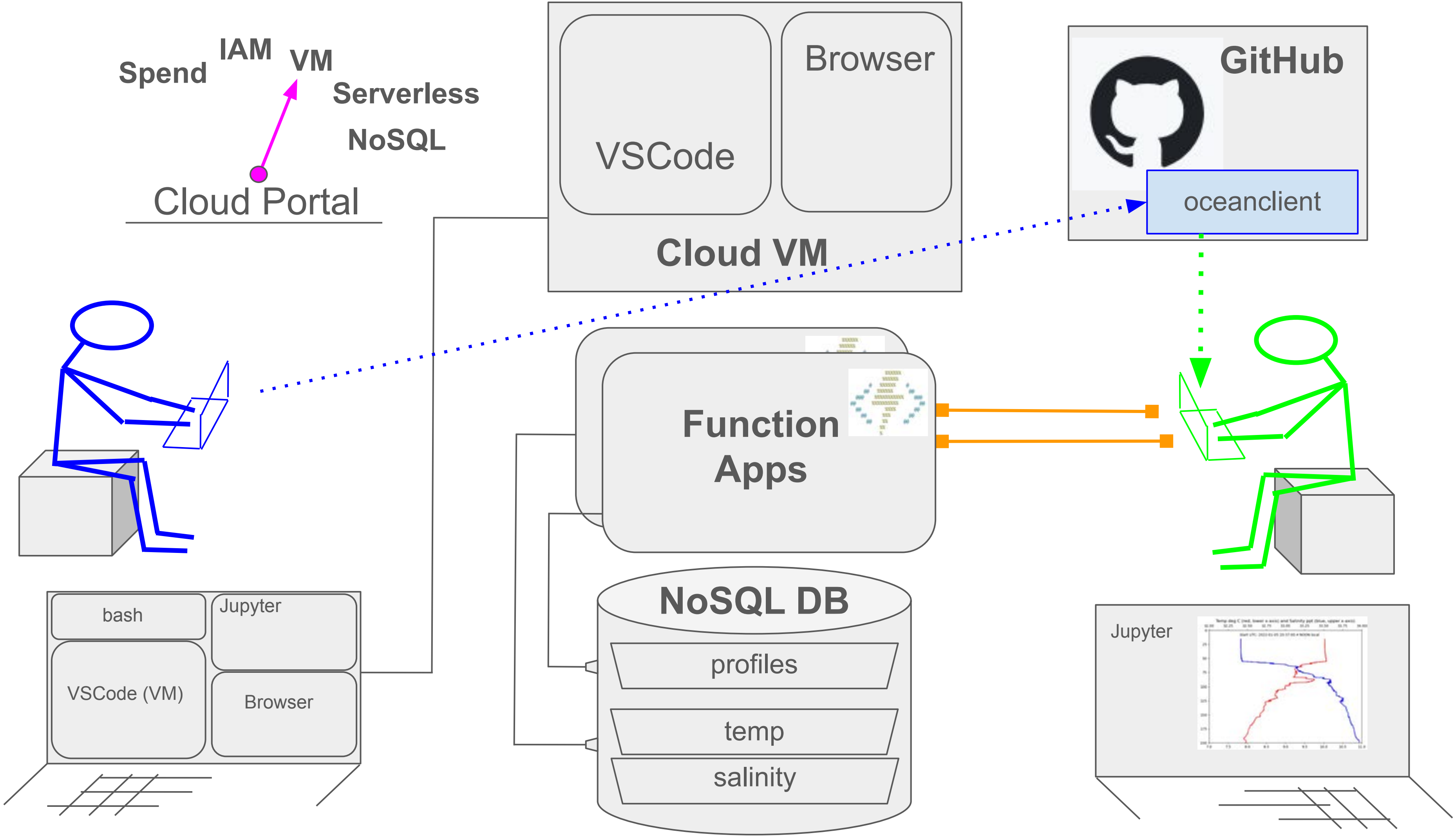
In English

The blue researcher/builder publishes both data and an access API to the cloud. This is open to everyone.

The researcher/builder next publishes an example Client on GitHub.

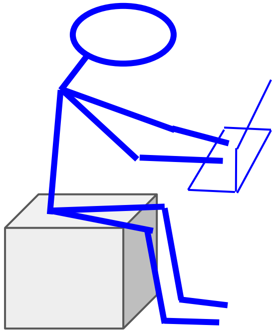
The green colleague downloads the Client and uses it to explore the data.

“Exploring the data” happens without needing to know how the system was built.



Spend
IAM
VM
Serverless
NoSQL
Cloud Portal
VSCode
Browser
Cloud VM
GitHub
oceanclient
Function
Apps
NoSQL DB
profiles
temp
salinity
bash
Jupyter
VSCode (VM)
Browser
Jupyter

Previous slide simplified



- My laptop
- Azure Portal
- VM (VSCode Server)
- Azure Functions
- Azure CosmosDB

I use the Naomi (MSE544) tutorial to orchestrate these resources; and then I follow the narrative for the periodic table.

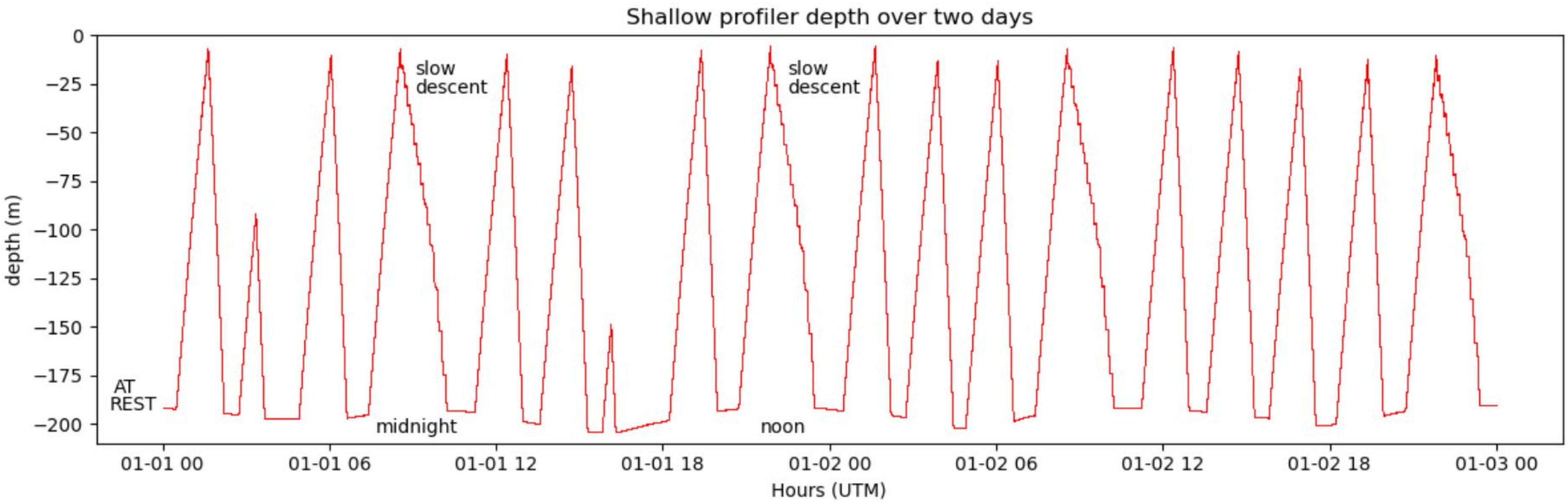
In the process I learn how the end result (a data API) is constructed from data in a NoSQL database wired up to a serverless function triggered by HTTP requests.

I have enough now to build my own Shoebox Gateway. I also build a custom Shoebox Client that I publish on GitHub.

Demo

Details…

Detail: Two days of profiler metadata



Detail: Why NoSQL?

- Actually everything here could be done in SQL
- Transactions were engineered to be safe during the advent of SQL
-

NoSQL

Link to NoSQL lecture notes

ACID view of database transactions

Atomicity: Transactions comprised of many statements are treated as single events

Consistency: Transactions move between *consistent* states of the database

Isolation: Analogous to linearity, in that multiple transactions proceeding asynchronous result in the same state as if they were executed sequentially

Durability: Completed transactions are not lost in the face of system failures such as power outage. Often implies a non-volatile memory component.

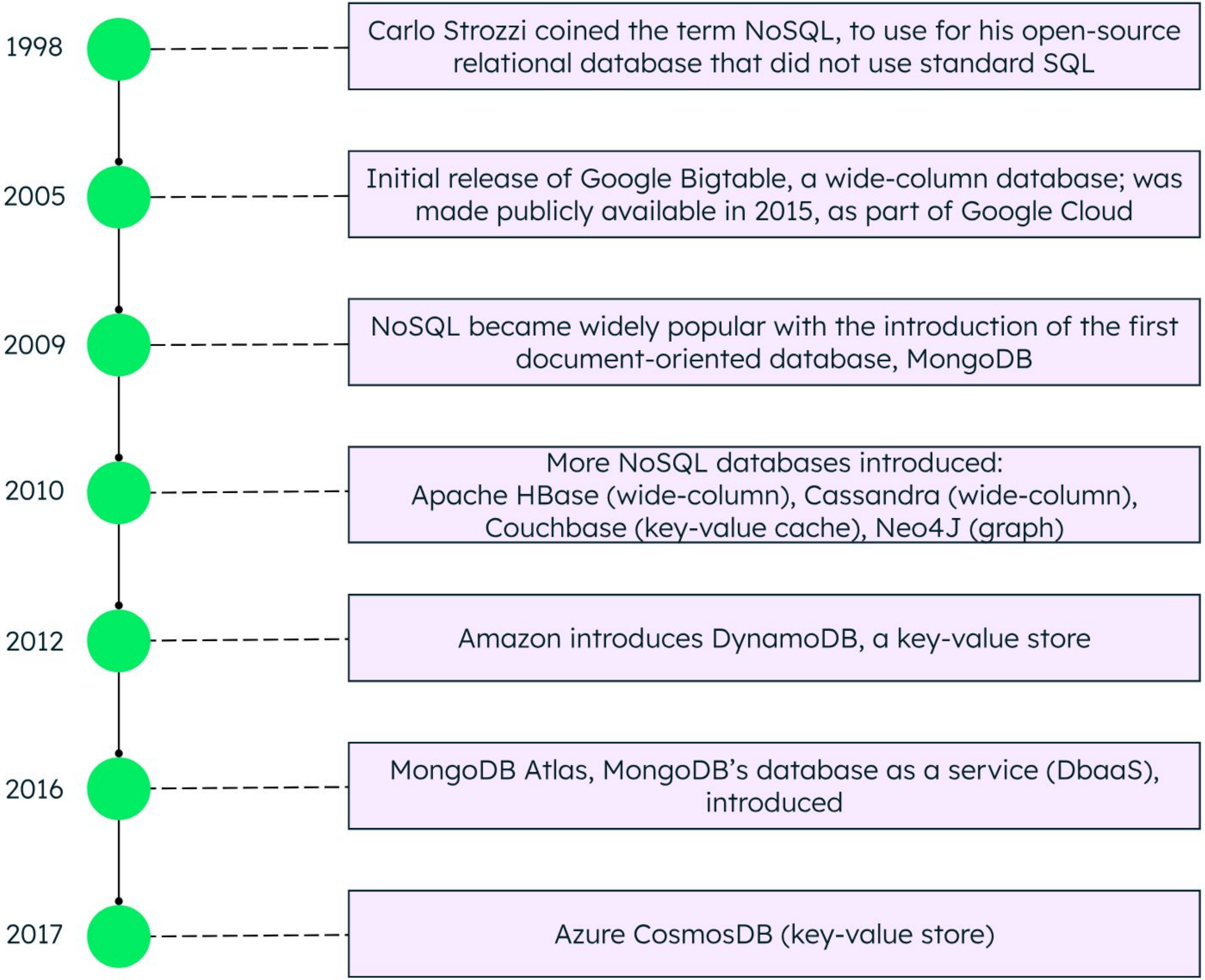
What does ACID mean for research data?

The event of interest is a *transaction* which changes the state of the database. Scientific data are subject to change. Derived data can be re-derived using new algorithms or otherwise modified or annotated. Sensor data over particular time intervals may be invalidated due to becoming uncalibrated. A time series may be augmented with new data products, for example water density inferred from temperature, salinity and pressure.

The ACID acronym calls out a set of desirable database attributes that ensure the data are available and won't be corrupted by colliding transactions and such.

A NoSQL Timeline

(more on NoSQL follows after we push through with a Goal 1 lightning tour: Periodic Table)



Detail: Testing the profile API in VSCode

http://localhost:7071/api/profile?day=30&index=9

Pretty-print ☑

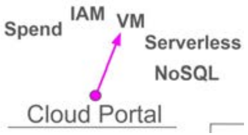

```
{
  "rest start time": "2022-01-30 23:25:00",
  "rest start depth": -192,
  "ascent start time": "2022-01-30 20:37:00",
  "ascent start depth": -191,
  "descent start time": "2022-01-30 21:49:00",
  "descent start depth": -13,
  "descent end time": "2022-01-31 02:08:00",
  "descent end depth": -189,
  "id": "2022-01-30 20:37:00"
}
```

Detail: Moving the profile API up to the big leagues

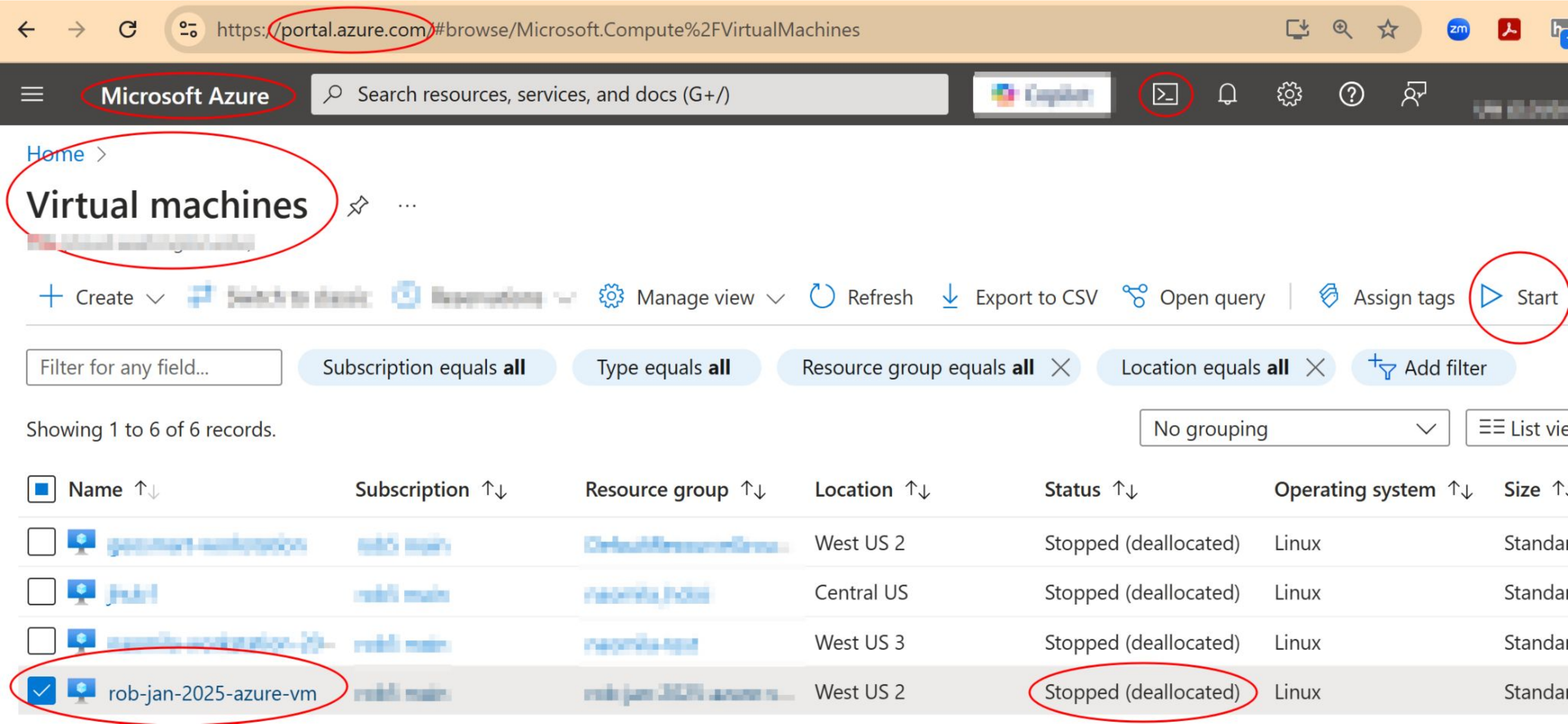
https://oceanography.azurewebsites.net/api/profile?day=17&index=9

Pretty-print

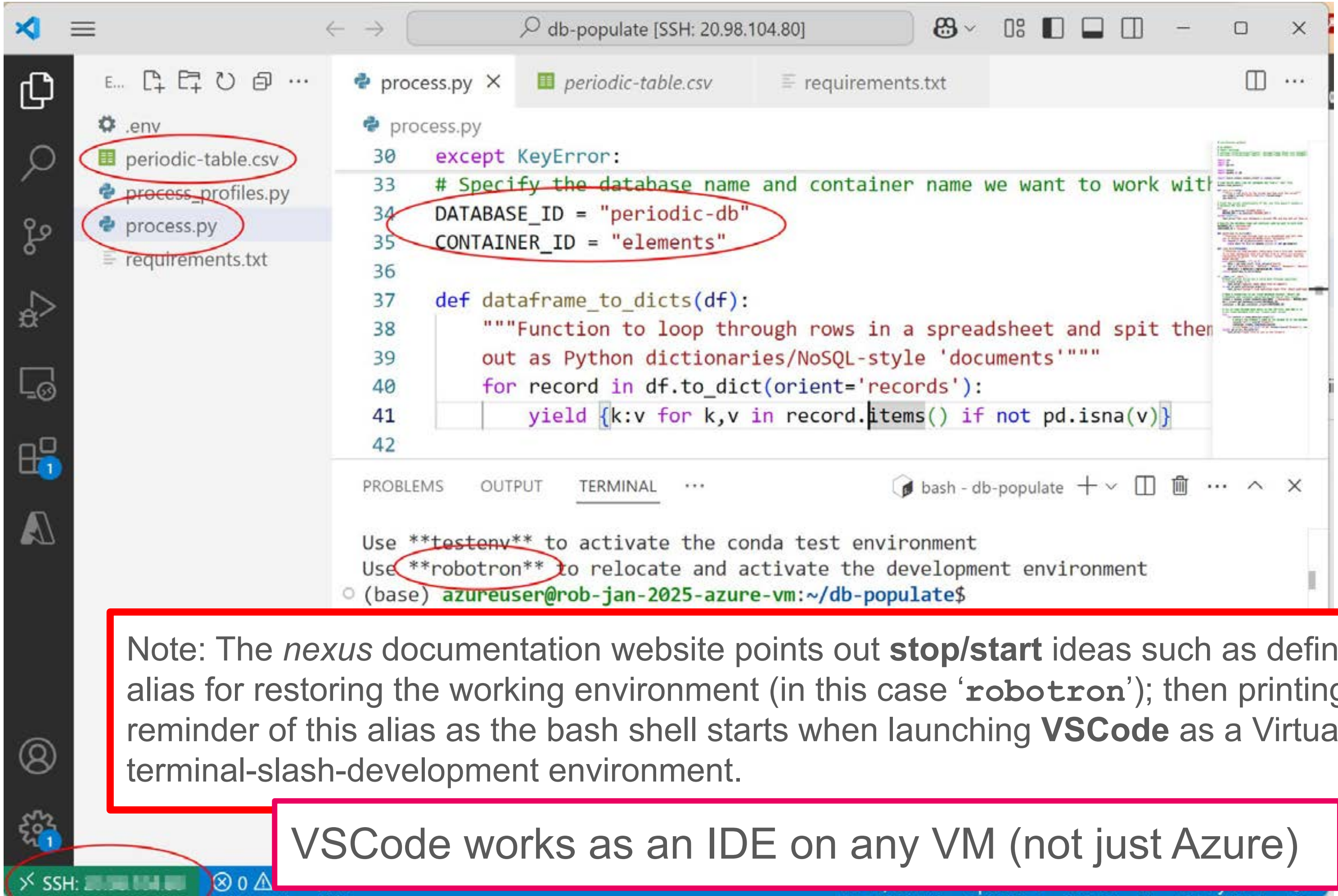

```
{
  "rest start time": "2022-01-17 23:26:00",
  "rest start depth": -190,
  "ascent start time": "2022-01-17 20:37:00",
  "ascent start depth": -193,
  "descent start time": "2022-01-17 21:49:00",
  "descent start depth": -13,
  "descent end time": "2022-01-18 02:09:00",
  "descent end depth": -189,
  "id": "2022-01-17 20:37:00"
}
```



Detail: The Azure portal in action



Detail: VSCode in action

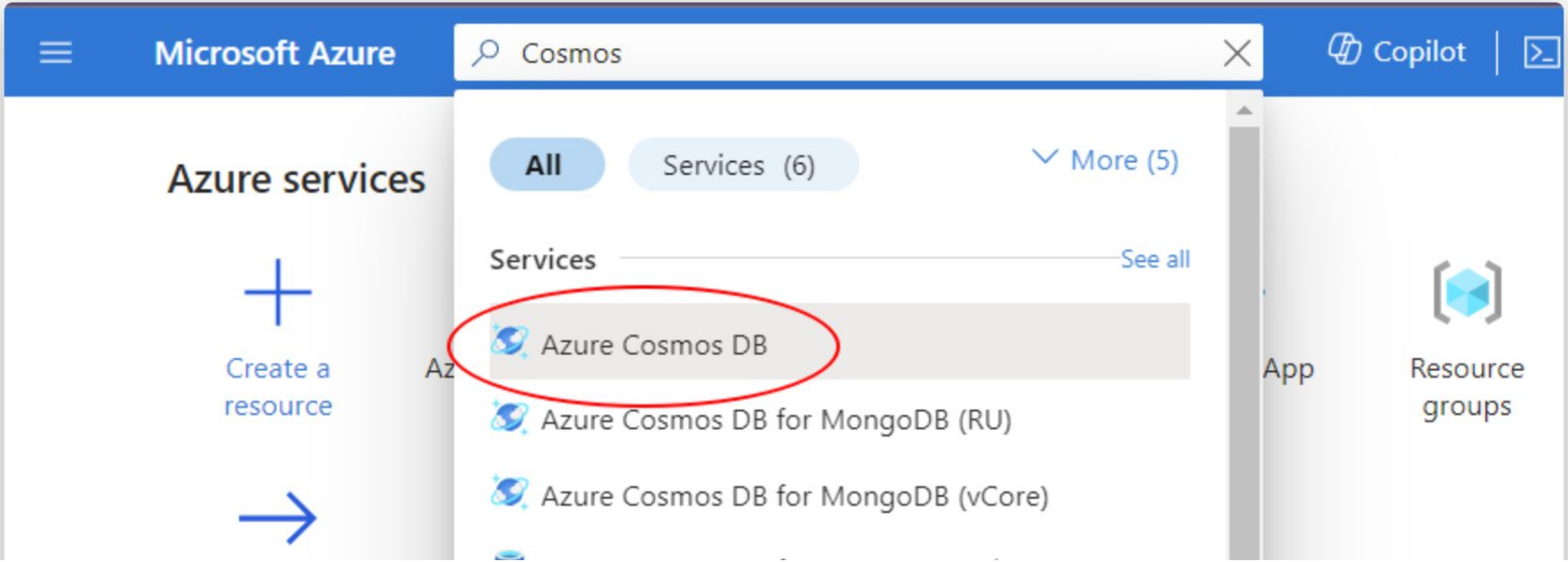


Note: The *nexus* documentation website points out **stop/start** ideas such as defining an alias for restoring the working environment (in this case '`robotron`'); then printing a reminder of this alias as the bash shell starts when launching **VSCode** as a Virtual Machine terminal-slash-development environment.

VSCode works as an IDE on any VM (not just Azure)

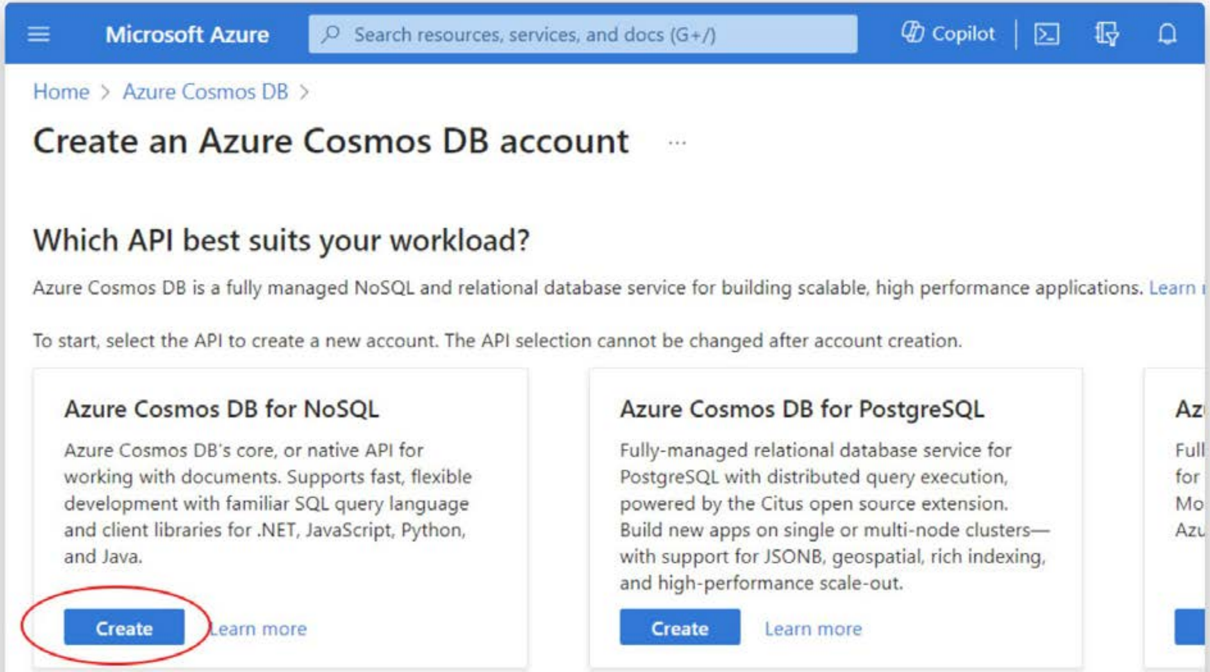
Detail: Azure Database Service

Go back to the web portal and search for `Cosmos`. Open up the dashboard for `Azure Cosmos DB`:



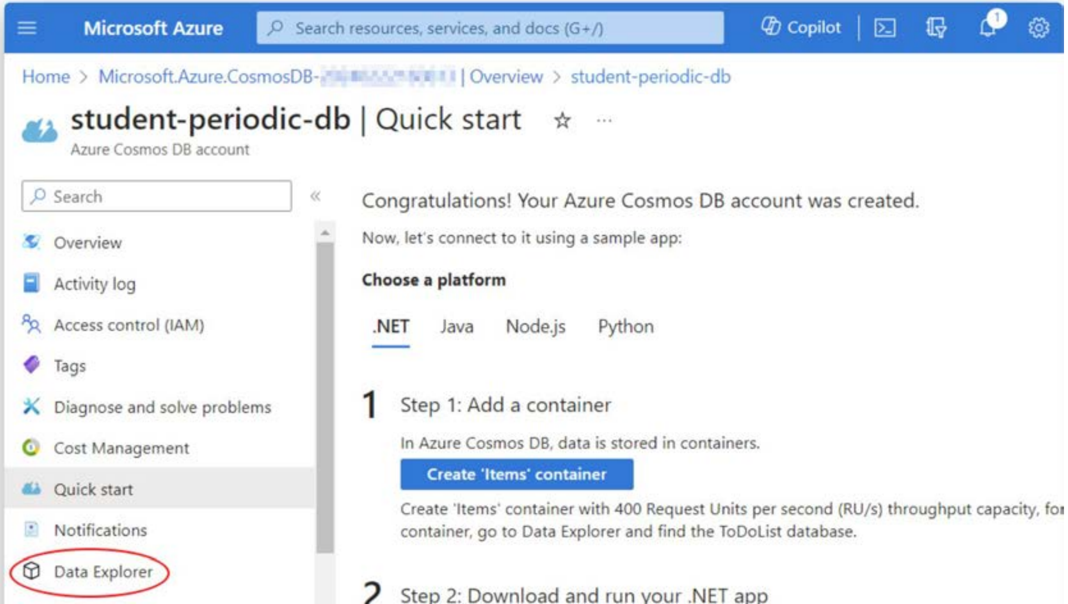
Detail: Creating a NoSQL Database in CosmosDB

Azure supports a number of different database technologies, all of which are provided with the brand name “Cosmos DB”. Today, we’ll be makind a NoSQL document store, which they call “Cosmos DB for NoSQL”. Click the `Create` button under the `Cosmos DB for NoSQL` heading:

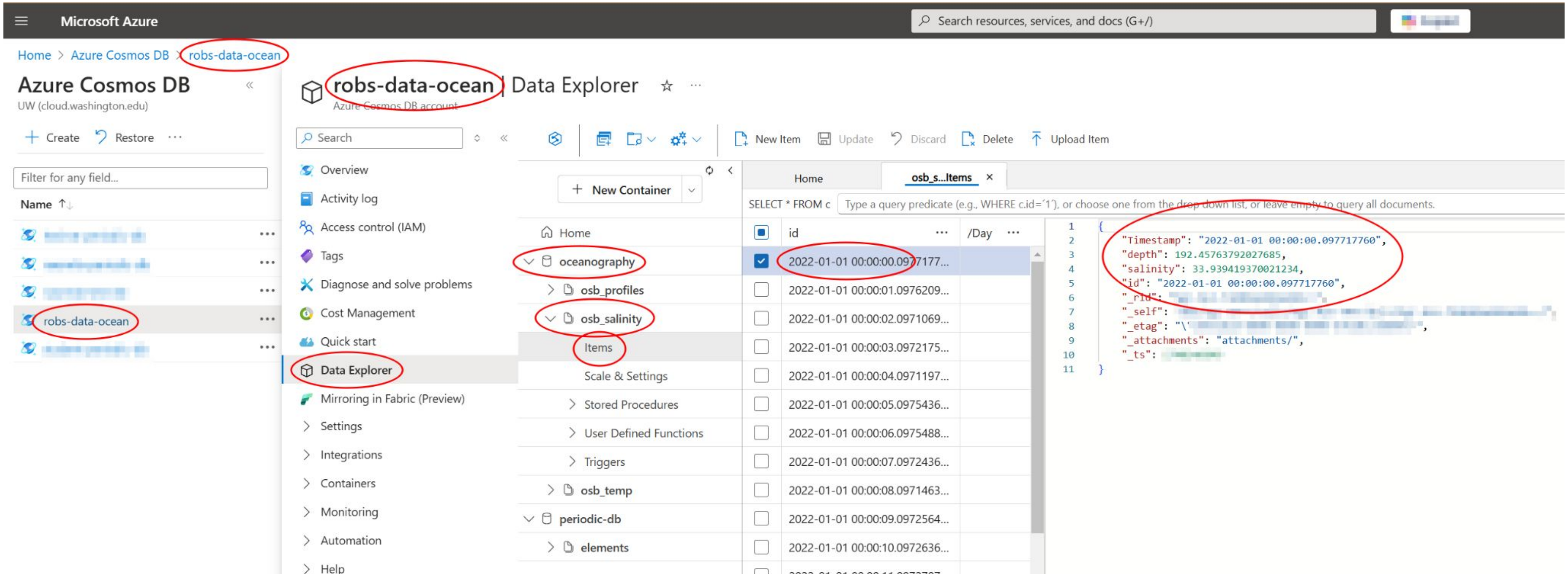


Detail: In-portal Data Explorer

The portal will bring us to a quickstart page, but we're not going to follow those instructions. Instead, select the `Data Explorer` option on the left:



Detail: Azure portal: Directed to the NoSQL database



Detail: API self-documenting

https://oceanography.azurewebsites.net/api/info

```
Oh Galaga!                    (info)

The 'profile' API has three necessary parameters:
  The route is 'profile' as in '/api/profile?'
  The first parameter is 'day', an integer from 1 to 31.
    This selects a day of the month of January 2022.
    Note: Sensor data are in place only for January 1 through 5
  The second parameter is 'index', an integer from 1 to 9.
    This selects one of the (up to) 9 profiles on that day.
      Profile 4 is local midnight. 9 is local noon.

      Example: <url>/api/profile?day=1&index=4

      Returns: Four profile timestamps
```

Detail: Simplest possible Python Client

```
import requests

r = requests.get("https://oceanography.azurewebsites.net/api/info")

print(r.text)
```

produces:

```
Oh Galaga!                              (help)

The 'profile' API has three necessary parameters:
  The route is 'profile' as in '/api/profile?'
  The first parameter is 'day', an integer from 1 to 31.
    This selects a day of the month of January 2022.
    Note: Sensor data are in place only for January 1 through 5
  The second parameter is 'index', an integer from 1 to 9.
    This selects one of the (up to) 9 profiles on that day.
      Profile 4 is local midnight. 9 is local noon.

      Example: <url>/api/profile?day=1&index=4

      Returns: Four profile timestamps
```

…same again but using the API to get a profile…

```
[6]: r = requests.get('https://oceanography.azurewebsites.net/api/profile?day=5&index=4')
     print(r.text.replace(",", ",\n"))
```


```
{"rest start time": "2022-01-05 08:27:00",
 "rest start depth": -196,
 "ascent start time": "2022-01-05 07:17:00",
 "ascent start depth": -194,
 "descent start time": "2022-01-05 07:54:00",
 "descent start depth": -109,
 "descent end time": "2022-01-05 12:56:00",
 "descent end depth": -193,
 "id": "2022-01-05 07:17:00"}
```

Detail: Self-testing???

Upon sober recursion I believe api testing is better done by a non-self.

Detail: More on NoSQL

MongoDB is the original open source NoSQL DBMS

https://www.mongodb.com/resources/basics/databases/nosql-explained

Conclusions

- Open science while not trivial is worth consideration of the effort investment
- Publishing data for open access is facilitated by cloud tech
  - API in serverless functions tapping in NoSQL
- This deck (to be made available) ties to other resources
  - The MSE544 “Serverless Azure Tutorial” by Naomi Alterman
  - Publishing / accessing cloud data: Periodic table and ocean observatory examples
- The CloudBank help desk is help@cloudbank.org

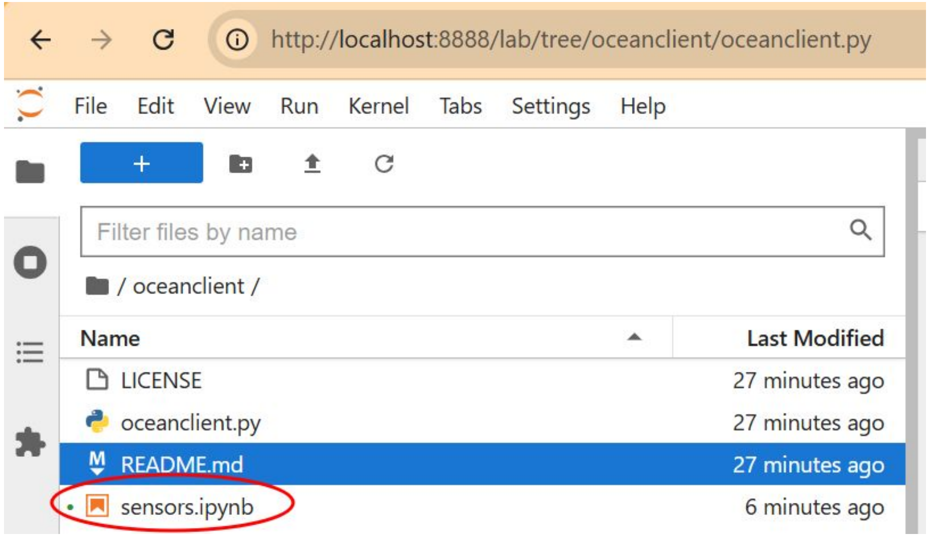
Thanks!

-The CloudBank team

https://docs.google.com/presentation/d/1qYIf3mTcfYtRY_I-zPkaqbIFLe2cHkuAIwnGnJpRQLc

4 Demo Backup Slides: What is supposed to happen

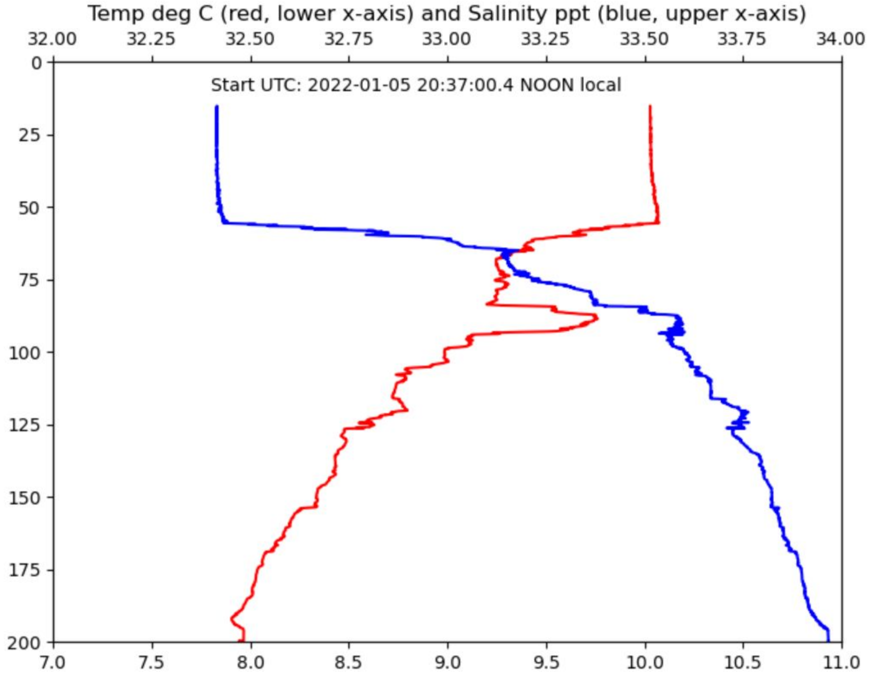
```
$ git clone https://github.com/robfatland/oceanclient
```



4 Demo Backup Slides: What is supposed to happen

```
[1]: import oceanclient as oc
     dfT, dfS = oc.Chart('2022-01-05', 9)
```

```
data query result type: <class 'list'> with 8760 elements
prep time 6.29 seconds; data vector length: 4380
```



4 Demo Backup Slides: What is supposed to happen

```
]: dfT
```

	Timestamp	depth	temp
0	2022-01-05 20:37:00.482559488	199.660778	7.943294
1	2022-01-05 20:37:01.482462720	199.662944	7.943356
2	2022-01-05 20:37:02.482989568	199.664009	7.943480
3	2022-01-05 20:37:03.482579456	199.659779	7.943480
4	2022-01-05 20:37:04.482899456	199.655482	7.943542
...	...	...	...
4375	2022-01-05 21:49:55.597626880	15.886139	10.028767
4376	2022-01-05 21:49:56.597320704	15.423047	10.028833
4377	2022-01-05 21:49:57.597327872	15.265456	10.028635
4378	2022-01-05 21:49:58.598376960	15.411262	10.028701
4379	2022-01-05 21:49:59.598070784	15.726319	10.028635

4380 rows × 3 columns

```
: dfS
```

	Timestamp	depth	salinity
0	2022-01-05 20:37:00.482559488	199.660778	33.967098
1	2022-01-05 20:37:01.482462720	199.662944	33.967234
2	2022-01-05 20:37:02.482989568	199.664009	33.967048
3	2022-01-05 20:37:03.482579456	199.659779	33.966984
4	2022-01-05 20:37:04.482899456	199.655482	33.966795
...	...	...	...
4375	2022-01-05 21:49:55.597626880	15.886139	32.415099
4376	2022-01-05 21:49:56.597320704	15.423047	32.414741
4377	2022-01-05 21:49:57.597327872	15.265456	32.414862
4378	2022-01-05 21:49:58.598376960	15.411262	32.414864
4379	2022-01-05 21:49:59.598070784	15.726319	32.414663

4380 rows × 3 columns

4 Demo Backup Slides: What is supposed to happen

```
def Chart(s, n):
    '''oceanclient.Chart(s, n) is hardwired into a demonstration data API. The source data is
    from the Regional Cabled Array program, Oregon Slope Base shallow profiler. The function
    returns two pandas Dataframes: one for temperature and one for salinity. It also creates
    a matplotlib chart of the data. Argument 's' is a date in January 2022 formatted as '2022-01-03'.
    Argument 'n' is an integer from 1 to 9 inclusive which is the profile index for that day.'''

    import requests, time, pandas as pd
    from numpy import datetime64 as dt64, timedelta64 as td64
    from matplotlib import pyplot as plt

    toc = time.time()

    # convert Chart() arguments to API-format strings
    day=str(int(s[8:10]))
    index=str(n)
```