Introduction to Pinecone indexes

VECTOR DATABASES FOR EMBEDDINGS WITH PINECONE

James Chapman
Curriculum Manager, DataCamp

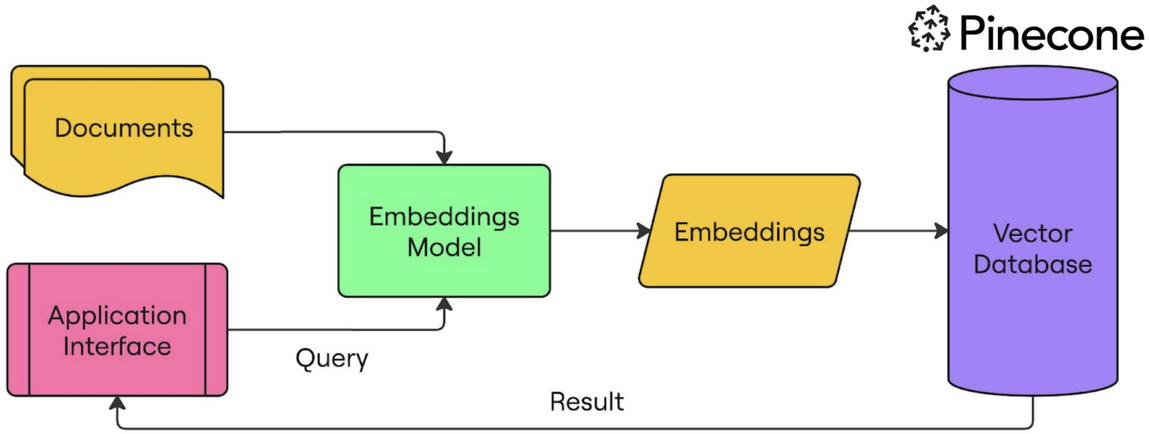
datacamp

Pinecone

→ Building and *scaling* **GenAI applications!**

You'll learn to...

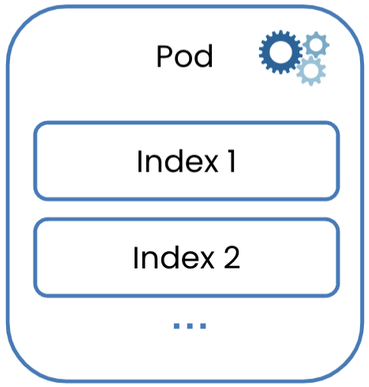
- Create indexes and ingest vectors
- Retrieve and query vectors
- Create common AI applications



Indexes

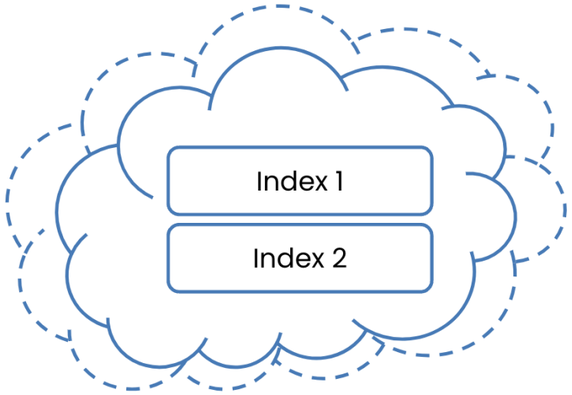
- **Store** vectors
- **Serve** queries and other vector manipulations
- Index contains **records** for each vector, including **metadata**
- Can create multiple indexes

Pod-Based



- Choose hardware to create the index → **pods**
- **Pod type** determines *storage*, query *latency*, query *throughout*

Serverless



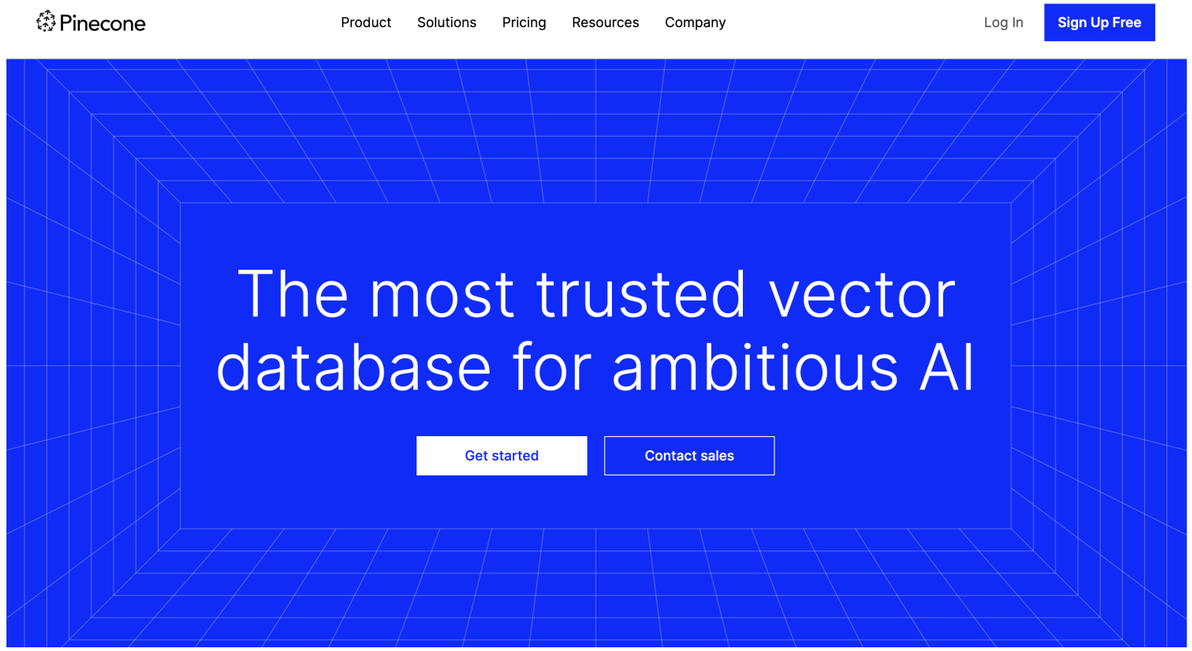
- No resource management
- Indexes *scale automatically*
- Run on *cloud* and store in blob
- Easier to use and often *lower cost*

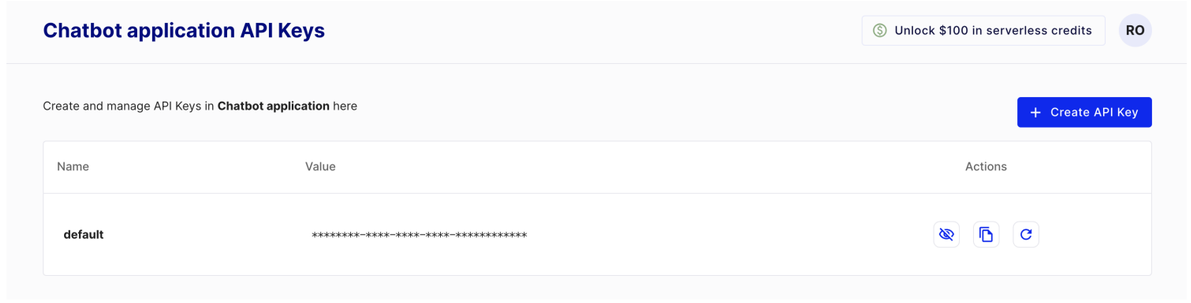
What we'll use in this course

[1] https://docs.pinecone.io/guides/indexes/understanding-indexes

Creating a Pinecone API key

1. Create a Pinecone Starter account → pinecone.io
2. Head to "API Keys"
3. Copy your API key





Creating a serverless index

```
from pinecone import Pinecone, ServerlessSpec

pc = Pinecone(api_key="API_KEY")

pc.create_index(
    name='datacamp-index',
    dimension=1536,
    spec=ServerlessSpec(
        cloud='aws',
        region='us-east-1'
    )
)
```

Checking our indexes

```
pc.list_indexes()
```

```
{'indexes': [{'dimension': 1536,
              'host': 'datacamp-index-1eef0f4.svc.aped-4627-b74a.pinecone.io',
              'metric': 'cosine',
              'name': 'datacamp-index',
              'spec': {'serverless': {'cloud': 'aws', 'region': 'us-east-1'}},
              'status': {'ready': True, 'state': 'Ready'}}]}
```

Let's practice!

VECTOR DATABASES FOR EMBEDDINGS WITH PINECONE

Managing indexes

VECTOR DATABASES FOR EMBEDDINGS WITH PINECONE

James Chapman
Curriculum Manager, DataCamp

datacamp

Connecting to the index

```
pc = Pinecone(api_key="API_KEY")

pc.create_index(
    name='datacamp-index',
    dimension=1536,
    spec=ServerlessSpec(
        cloud='aws',
        region='us-east-1'
    )
)

index = pc.Index('datacamp-index')
```

Connecting to the index

```
index = pc.Index('datacamp-first')
```

```
...
pinecone.core.client.exceptions.NotFoundException: (404)
Reason: Not Found
HTTP response headers: HTTPHeaderDict({'content-type': 'text/plain; charset=...
HTTP response body: {"error":{"code":"NOT_FOUND","message":"Resource datacamp-first
not found"},"status":404}
```

Index statistics

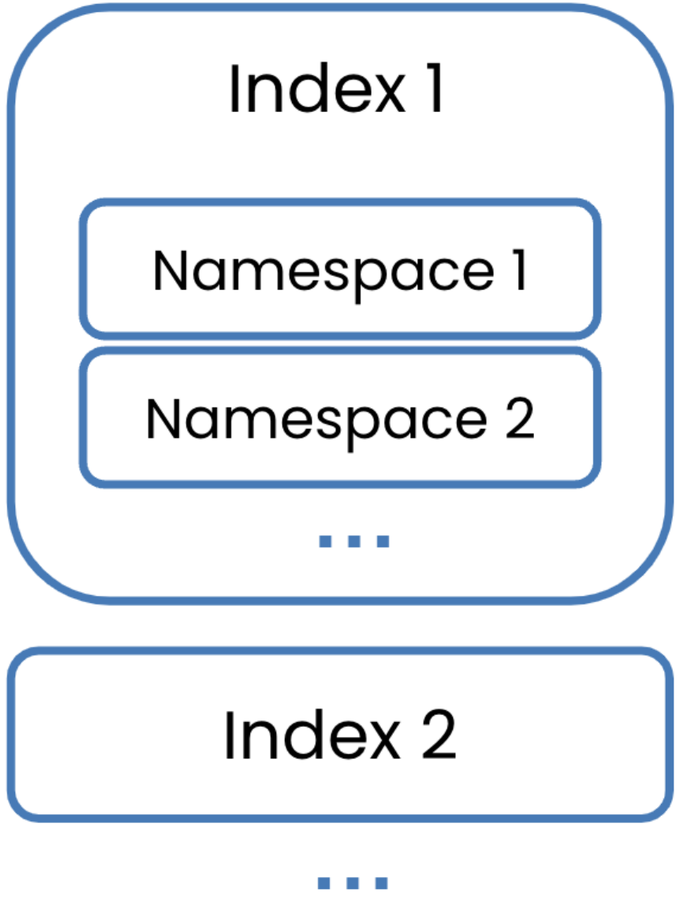
```
index.describe_index_stats()
```

```
{'dimension': 1536,
 'index_fullness': 0.0,
 'namespaces': {},
 'total_vector_count': 0}
```

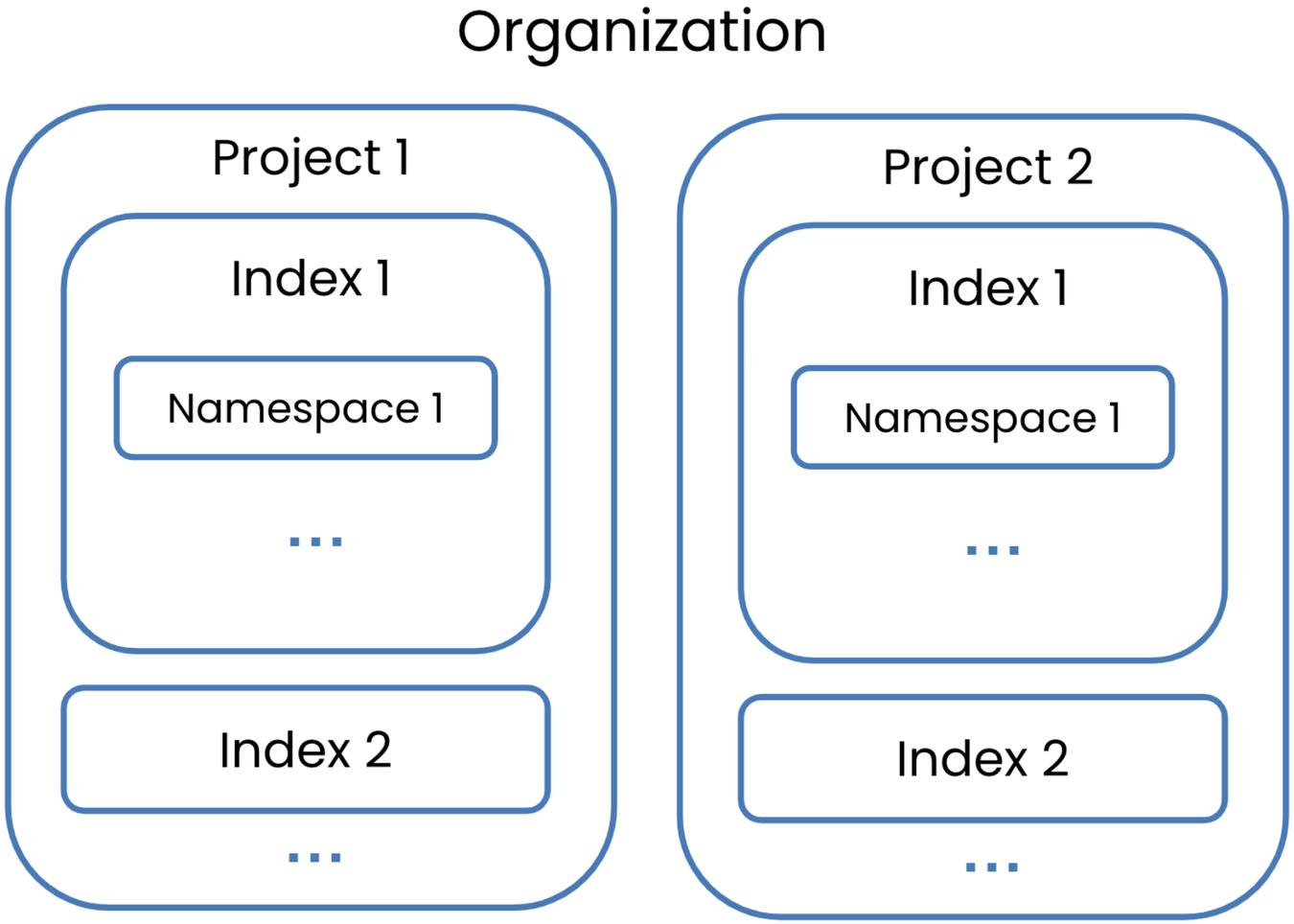
Namespaces

- Containers for *partitioning* indexes
  - Separate datasets
  - Data versioning
  - Separate groups

Focus on the single namespace case for now



Organizations



Organizations

Organization Owner

- Permissions across entire org.
- Manage billing, users, *all* projects

Organization User

- *Restricted* org-level permissions
- Invited to *specific projects*
- Become owner to those projects

Deleting indexes

```
pc.delete_index('datacamp-index')


pc.list_indexes()
```

```
{'indexes': []}
```

Let's practice!

VECTOR DATABASES FOR EMBEDDINGS WITH PINECONE

Vector ingestion

VECTOR DATABASES FOR EMBEDDINGS WITH PINECONE

James Chapman
Curriculum Manager, DataCamp

datacamp

Creating and connecting to an index

```
pc = Pinecone(api_key="API_KEY")

pc.create_index(
    name='datacamp-index',
    dimension=1536,
    spec=ServerlessSpec(
        cloud='aws',
        region='us-east-1'
    )
)

index = pc.Index('datacamp-index')
```

Ingesting vectors

```
vectors = [
    {
        "id": "0",
        "values": [0.0255255547564029694, ..., 0.0188823901116848]
    },
        ...,
    {
        "id": "9",
        "values": [0.0207124687731266O2, ..., 0.0064184423536062 24]
    },
]
```

Checking dimensionality

```
vector_dims = [len(vector['values']) == 1536 for vector in vectors]
all(vector_dims)
```

```
True
```

```
PineconeApiException: (400)
Reason: Bad Request
HTTP response headers: HTTPHeaderDict({'Date': 'Fri, 17 May 2024 10:54:57 GMT', ...
HTTP response body: {"code":3,"message":"Vector dimension 256 does not match the
dimension of the index 1536","details":[]}
```

Upserting vectors

- `.upsert()` : Update or insert

```
index.upsert(
    vectors=vectors
)

index.describe_index_stats()
```

```
{'dimension': 1536,
 'index_fullness': 0.0,
 'namespaces': {'': {'vector_count': 10}},
 'total_vector_count': 10}
```

Ingesting vectors with metadata

```
vectors = [
    {
        "id": "0",
        "values": [0.025525547564029694, ..., 0.0188823901116848]
        "metadata": {"genre": "productivity", "year": 2020}
    },
        ...,
]
```

- **Metadata:** data about data!
  - Can be utilized for *metadata filtering* → **Chapter 2**

Upserting vectors with metadata

```
index.upsert(
    vectors=vectors
)
```

Remember to use the structure:

```
    {
        "id": "0",
        "values": [0.025525547564029694, ..., 0.0188823901116848]
        "metadata": {"genre": "productivity", "year": 2020}
    },
        ...,
```

Let's practice!

VECTOR DATABASES FOR EMBEDDINGS WITH PINECONE