Practical approaches for web scraping for research – using Airbnb as an example data provider

Transcript from webinar video recording

1
00:00:00,018 --> 00:00:01,526
First of all, welcome everybody

2
00:00:01,626 --> 00:00:05,074
to the latest in a series of webinars

3
00:00:05,174 --> 00:00:09,241
being presented by the Urban Big Data Centre.

4
00:00:09,739 --> 00:00:14,606
This session, hopefully you’re not surprised to see, is entitled

5
00:00:14,706 --> 00:00:16,885
Practical Approaches for Web Scraping
using Airbnb data and the UBDC Airbnb scraping platform.

My name is Andrew McHugh.

I'm the Senior Data Science Manager at the Urban Big Data Centre and I will describe what we do as a centre in the following slides.

But I'll ask my colleague, Nick Ves, who is the architect of the platform we'll showcase today, to introduce himself just now as well.
Hi, good morning all.

I am a Data Scientist working with the UBDC and Andrew here.

And it’s a pleasure that you guys have come here.

We are going to present to you, later on,

the way that we have been using to capture data from Airbnb.

So, hopefully you will all enjoy it and welcome.

Thanks, Nick. So, we’ve got
a relatively packed agenda today,

but I want to start with some housekeeping.

This session is being recorded,

as you will no doubt have been notified when you joined the Zoom session.

So, an edited video will be published on

UBDC's YouTube Channel in due course.

That's the YouTube link where you'll find

all Urban Big Data Centre video content.
But just, obviously, to bear in mind that that is happening.

All slides that we will present today will be made available to course participants, as well as the code that will form the major part of today's focus.

Also all accessible.

Questions can and we encourage you to send those in via the Zoom chat.

We will have time at the end
and we'll also take some time in the middle of the session to cover those.

But they will be monitored throughout, so if you have some more technical questions or bigger questions, we'll tackle those at the appropriate times.

And we ask, obviously, that people keep mics muted throughout the session. And then the principal form of interactivity will be
through the Zoom chat.

So, just to summarise what we'll be covering today.

You'll have seen the intended learning outcomes already, perhaps.

But following today's webinar,

we hope you should be able to accomplish a number of things.

And the main focus, the principle focus today is

UBDC's open-source platform for scraping Airbnb data.

So, you'll be able to
describe the installation, configuration, and deployment of that.

And later on, having grabbed the code for yourself,

you'll be able to recreate that within your own environment.

You'll be able to explain what web scraping means in very broad terms.

Understand what the key legal, ethical, and data governance issues associated with web scraping are.

Summarise the limits on data use and sharing.

To identify and select appropriate datasets for web scraping.
Understand what an API is.

Learn how to negotiate some of the technical barriers that we associate with web scraping.

And lastly, to systematise scale and optimise your approaches.

So, as I say, this will be principally through the lens of our own implementation, our own scraping platform.

But hopefully the lessons and the learnings
that we describe will be

68
00:03:51,896 --> 00:03:54,935
applicable to other kinds of problems that you may be tackling.

69
00:03:55,774 --> 00:03:58,083
In terms of the programme, we're here for two hours today.

70
00:03:59,642 --> 00:04:05,173
We're in the first of those agenda items at the moment.

71
00:04:06,389 --> 00:04:08,966
We will have a talk that follows.

72
00:04:09,066 --> 00:04:13,179
I will lead you through some of the background to this work.

73
00:04:13,351 --> 00:04:17,790
before handing over to Nick for the main part of the agenda,

74
00:04:17,890 --> 00:04:20,725
which will be a technical discussion of
how we've approached the scraping problem

and how we've implemented a platform.

And latterly, of course, how that platform works

and how you can use it yourself.

And, as I said, again,

encouraging as much discussion and questions as possible

to try and ensure that this answers any of those questions that may arise

or are already on your mind.
Before we get started, I wanted to draw your attention to just one other webinar. I mentioned that this is the latest in a series of UBDC webinars.

We have another one, later this week, on Thursday, the 24th June at 10am,

which is something of a companion piece to this one.

Whereas we focus today on how you go about accumulating collections of
data through web scraping,

we'll have a companion session on Thursday

led by my colleagues Mark Livingston and Yang Wang

and they will be focusing much more on

how you actually use the Airbnb web scraped data

to undertake some pertinent research questions

and research challenges.

So, that takes place on Thursday.
You can visit the UBDC website for more details and register.

And, again, like all of these webinars, that's free to participate in.

So, moving on, I'll offer just a wee summary of what the Urban Big Data Centre does because, again, it may give you some context and it may make some of what follows a bit clearer.

We're based at the University of Glasgow in the United Kingdom.
We are funded by the Economic and Social Research Council, which is a UK government funder of social science research and associated infrastructure.

As well as receiving some funding from the University of Glasgow itself.

We've got several priorities, which are associated with two principal functions.

So, we operate a traditional research centre focused on better understanding cities.
and relying on the use of new and emerging forms of data

and associated methods

to enhance our understanding of

how cities function and how their residents behave and interact.

And we also offer, on behalf of

the Economic and Social Research Council,

a national data service where we support other, mainly academic, researchers
in exploring similar themes in similar kinds of ways.

But amid that, we have these priorities around data infrastructure and collections.

So, we're actively developing our data infrastructure and associated data collections.

We support a number of priority research strands within the Centre, although not exclusively for those that we support externally.
But those include things like transport and mobility;

neighbourhood, housing, and environment;

education, skills, and productivity;

and some of the more umbrella questions, if you like,

around the role and value of big data and urban governance.

And our aims are to achieve public policy impact,

to critically evaluate the role and the value of

big data and urban analytics,
and to enhance the data collections that we create and collect

and the associated methods for their use.

And all of these various different functions and priorities often come back to or rely upon the availability of data.

And getting hold of data is, as you can imagine, and no doubt many of you have experienced, can be quite a challenge at times.

It takes up a lot of our time
to go out and not only identify data

but to develop the means by which we can get hold of that data.

And that can imply a range of technical challenges

as well as, obviously, legal challenges and so on.

And obviously there are many reasons why organisations don't

make their data available

or do not feel incentivised to do so.
Including a lack of capacity to do so;

perceptions of risk;

perceived conflict with existing business models

if, for example, making data available may undermine

the business models which involve the provision of data

on a commercial basis

or other kinds of conflicts;

and fears too that research results may reflect badly on them
or expose them to criticism or be received negatively by other stakeholders.

That last point is probably one that we focus on most in the content that follows and in our particular case.

And while there are many reasons why we feel we can encourage organisations to make their data available, they don't always work.

So, sometimes we have to find other means of getting hold of data.
that aren't quite as collaborative as, for example, a licensing agreement or a collaboration agreement.

So, Airbnb is obviously the focus of today's session. It provides the lens through which we'll look at the technology discussion that will follow later on.

So, I thought it would be useful, again, to give you a little bit of a summary of what we were looking for and for what reason.

I'll start with a very, very fundamental point.
So, many of you will no doubt be familiar with Airbnb and no doubt have used their services in the past yourself.

But it’s an online marketplace for arranging or offering lodgings, homestays, and tourism experiences.

So, you can onto Airbnb and perform one of several roles.

The most obvious ones being there are hosts, there are people that own properties.
or that manage properties on behalf of others who use Airbnb as a platform

to promote the availability of these homes or lodgings

for tourism or for other kinds of accommodation purposes.

Meanwhile, there is another set of people

who are on Airbnb from a consumer point of view.

People who are interested in finding a holiday home,

finding somewhere to stay for a short-term basis.

And they will discover the available Airbnb homes
or venues or accommodations via that platform.

Our researchers were looking at Airbnb and were particularly interested in some of the impacts of Airbnb and the rapidly growing sharing economy that it’s a really important manifestation of.

What are the impacts of that?

What’s the importance of that for the private rented sector property market?

How does this impact on more traditional private rental?
We hear of these things, obviously as being very disruptive and we wanted to be able to characterise and understand what those disruptive impacts actually are.

Question two, the extent to which they are positive or potentially damaging. Some of the questions that are foremost in our researchers' minds.

The impact that Airbnb has on the availability of private rented stock.
So, given the opportunity to market and sell properties or rather sell access to properties on Airbnb, does that limit the amount of private rental stock that's available on a more traditional basis? Is there a spatial aspect? Are there particular locations or areas that see a greater or lesser impact of these changes?
What, if any, is the relationship with areas of social deprivation?

And does the rise of the sharing economy increase things like inner city gentrification and the suburbanisation of poverty?

So, things we would often associate with negative outcomes.

Are Airbnb properties falling below the standards for the private rented sector?

E.g. in terms of occupancy levels.
So, where there is a regulated environment,

are there standards that are not being met?

Has this been providing a loophole for substandard service provision?

And what impact, if any,

does Airbnb have on the existing hospitality industry,

as well as private rented sectors?

So, as you can no doubt imagine...

And I won’t go into these in any more detail than that.
you'll see perhaps some more coverage of some of these specific research questions and how they are tackled and resolved in the later webinar on Thursday.

But these are really just presented to show that these are not necessarily research interests or goals that will resonate with Airbnb as being something they are keen to get involved in. They may not be enthusiastic about facilitating such research, which may have the potential to
lead to outcomes which may be critical of their business models

and so on.

But given their possible lack of enthusiasm about making data available,

we have to look to sources elsewhere.

Unfortunately, as an unregulated sector,

there is little reliable data available

so we have to think creatively about how we can get the data
that will give us the insights into how this model works

00:13:56,415 --> 00:13:58,033
and what its implications are.

00:13:59,424 --> 00:14:02,341
So, we wanted to automate the scraping of Airbnb.

00:14:03,061 --> 00:14:07,127
As an online marketplace, the data is available on the web,

00:14:07,336 --> 00:14:09,795
it's accessible freely.

00:14:10,882 --> 00:14:15,181
And we wanted to effectively industrialise the collection of data

00:14:15,281 --> 00:14:18,000
from that platform on a very systematic basis

00:14:18,917 --> 00:14:21,218
in order to facilitate our research.
An obvious initial starting point was the legality of our proposed data collection.

To what extent are we legally entitled or permitted to go and collect data at scale from Airbnb in that kind of way?

I'll cover that in some slides in a moment or so.

But then, if we're able to reassure ourselves that we are legally permitted, we have a sound legal basis
for proceeding in that particular way,

can we then develop methods and can we design our data collection to support questions around the scale and growth of Airbnb?

Spatial change?

Focus on short-term lets?

So, can we see the changes manifest on a spatial basis?

Can we understand how occupancy works?

How many people are occupying particular Airbnbs at a given time?
Can we understand price and the changes in price?

And what some of the determinants of price and the things that influence price are?

And availability as well.

So, these are the things that we may feel are self-evident within the data, to some extent,
we feel we can sort of infer some of these things on a limited basis.

277
00:15:45,087 --> 00:15:47,005
But can we capture these at scale

278
00:15:47,105 --> 00:15:53,062
and achieve a robust and a credible understanding of some of these things?

279
00:15:54,603 --> 00:15:58,231
So, that will frame how we actually go about

280
00:15:58,331 --> 00:16:00,559
doing the scraping, which we'll see later on.

281
00:16:00,659 --> 00:16:02,812
But, again, before going there,

282
00:16:02,912 --> 00:16:08,220
I think it's important just to cover some of the fundamentals.

283
00:16:08,766 --> 00:16:12,384
And the first is just to offer a definition of scraping itself.
So, I've already mentioned in the past that we'd go out and get data from a variety of organisations in support of our research and very often that can be a very collaborative thing. It can be a very explicitly consensual thing. Things like license agreements and so on. Scraping is, comparatively, a one-sided activity. It's about taking data that exists, for example, on a website, and employing any one or a combination of
a range of different technical means
to automate the transfer of data from that location or resource.
And there's a whole range of methods that can be done to facilitate that.
So, with online data, we can employ techniques like text pattern matching
to grab particular types of data
and to then impose some kind of
structure on those at the collection end.
We can use things like HTTP programming.
And that will be the primary focus of what we do, relying on the Airbnb API, as Nick will outline later on.

We can rely on the existing structure that is implicit within web content through HTML and the Document Object Models and effectively parse through marked up content to extract particular data, and then obviously impose structure.
or transpose from the structure already evident.

And we can use a whole range of other more sophisticated techniques too,

like computer vision, for instance,

where we're actually analysing the visual representation of

web content in order to extract particular parts.

What's, I think, true of all of these approaches

and is core to the technical philosophy of this approach is that

it's unsupported by a web content owner.
The very fact that you're doing this is probably because
the web content owner is resistant or unable to make the data available under a cooperative model.
That means that while it may work very well one day, it might be prone to break in the event of things like changes to websites.
Web scraping technologies can fall down.
They are quite comparatively brittle in that respect.

And there can be a whole range of different website controls in place to limit one's ability to do things like scraping. Those can include common controls that are implicit in many websites that are designed to block nuisance hosts. So, when you start doing things at scale,
when you’re accumulating a lot of data,

then the behaviours, as far as the websites are concerned,

may look like a nuisance host,

like behaviours akin to denial of service attacks.

You need a lot of requests in a very short space of time

and there can be controls in place to blacklist nuisance clients

or those identified as nuisance clients.
And, obviously, as well as technical measures like those,

there can also be contractual measures.

The terms and conditions of websites that explicitly prohibit people from doing web scraping activities.

And as we'll see,

there may be those explicit contractual measures

but, likewise, there are other parts of the law

that offer us some encouragement and some reassurance
that those contractual measures may not be applicable.

We'll cover how we resolve all of these types of issues in the slides that follow.

There are some alternatives to scraping, of course, and even without the active participation of Airbnb themselves.

There are some existing online resources that we have reviewed and assessed.
Two prominent data providers are Inside Airbnb and AirDNA.

The former is mostly the fruits of data collection efforts undertaken by a small number of people who focused on particular geographic areas and particular prominent global cities and sought to build a picture using a range of different scraping tools.

AirDNA is a commercial platform.
where they will actually provide access to data at commercial cost.

Our view, having reviewed these,

is that they do tend to suffer from a number of shortcomings,

principally around things like limited sampling.

So, for instance, whilst Inside Airbnb offers

interesting data and it's been a useful means of

calibrating and validating some of our data collection efforts,
the sampling is quite limited to particular specific global cities

which don't necessarily intersect with those areas of interest that we have within our research agenda.

There are data quality concerns at times and also, I think, with AirDNA specifically,

there was uncertainty at times of exactly how data was being collected and an issue of black box processing,
which, from a research perspective, is particularly unattractive because we want to have a degree of reassurance that the data we're getting and the calculated variables or the inferred variables that are being presented within data sets, we have a good understanding of how those are arrived at. In particular, our understanding of occupancy, price, and availability, which are very key to our interests, it was not clear how the data related to those types of features was
arrived at in the AirDNA platform.

So, our view was that we'd be much more comfortable with developing our own data collection approach that much more directly relates to and aligns with the research needs that we had identified.

So, how do you begin to develop a scraping approach?

Well, there are various open-source projects out there that we initially evaluated.
And, for example, with the Inside Airbnb platform,

there are links to corresponding code bases that,

where effectively you can deploy

their tool sets in different locations on different schedules.

But we’ve reviewed a number of...

And that's the first link here, the Airbnb data collection project.

But for each of those that we reviewed, we determined that there were problems
and challenges with, for example, code bases that had not been updated,

that we were struggling to deploy successfully,

and we became increasingly aware of

a different approach that we were keen to adopt.

So, these projects here...

Well, the first link here, Airbnb data collection,

it's grabbing data directly using the Document Object Model, I think.

The latter one here, HTTPs requests randomiser is
a project that allows you to effectively distribute requests

across a number of different proxy hosts

in order to not fall foul of some of those website controls

I described earlier on.

We've devised our own alternative approach

that covers both of these things that Nick will be able to explain

much more comprehensively than I can.

But, again, we mentioned the issue of legal and ethical issues.
And, again, it's important to resolve these

and to feel reassured that

we're not falling foul of ethical or legal expectations

in order to undertake this kind of data collection work.

Unfortunately, despite being hopeful there would be a very clear

and unambiguous legal basis or lawful basis for us to proceed,

we're still very much involved in

a broader effort to achieve clarity there
that's probably going to prove elusive for a little bit longer.

So, we're based at the University of Glasgow and we worked with the CREATe centre, which is also based at the University of Glasgow, to better understand the legal landscape that relates to online scraping.

As I said, unfortunately the picture is not wholly clear. So, we were initially alerted to something within copyright law.
called the "text and data mining" exception,

which effectively, from a lay person's perspective,

provides a provision whereby for academic research

one is permitted to collect at scale

data that is made publicly available,

that is covered by copyright but is nonetheless made publicly available.

So, you have legal access to that.

And under this particular exception,
you are lawfully permitted to collect that data.

and to use it as the basis of academic research work.

Now, that is a right that cannot simply be ruled out through the use of contracts and terms and conditions.

So, it’s not possible to contract your way out of that situation from Airbnb’s point of view.

So, we learned of this and took this to CREATe and felt relatively reassured that
this would provide us our lawful basis to proceed.

But unfortunately, as I said, the issue is not quite as clear as we'd hoped because, in addition to copyright law, there are a range of other different legal regimes including privacy, including contracts, and confidentiality and other intellectual property rights too which themselves appear to be
operating in a somewhat contradictory fashion.

So, although we can look at copyright law and feel relatively reassured,

there may be other relevant legal precedents from other legal regimes that would cast some doubt in terms of the lawfulness of what we wish to do.

Ultimately, until case law emerges,
this therefore remains a risk-based policy decision.

463
00:27:19,972 --> 00:27:23,475
Do you do this on the basis of a risk-based decision.

464
00:27:23,575 --> 00:27:25,651
And some of the factors that we've considered

465
00:27:25,751 --> 00:27:28,015
and felt relatively reassured about,

466
00:27:28,823 --> 00:27:33,387
having reviewed in great detail the case law that already exists,

467
00:27:33,487 --> 00:27:35,153
there's very little if anything

468
00:27:35,253 --> 00:27:38,011
that feels relevant to our particular setting.

469
00:27:38,111 --> 00:27:40,556
It's obviously from a lot of different commercial environments
and clearly our agenda is not to achieve some kind of commercial outcome,

rather we’re seeking to deliver academic research that will lead to, that will be influential to policy, perhaps,

but without any kind of commercial expectations.

Similarly, we looked at other academics who were doing similar work and feel relatively reassured that there is an emerging code of practice around how one approaches this stuff responsibly.
The work that we did with CREATe has been written up into an initial working paper, which is linked here and you can see a DOI, and I would encourage people to familiarise themselves with the law of data scraping. And it's work that we'll very much continue to develop because there is a really important part here. We'll talk about some of the ways that
Nick has innovated in very technical terms to achieve a means of data collection which will hopefully be the basis of some quite transformative research.

But the innovation space here is not limited just to the technology, of course.

And within the legal environment, there is a great deal of opportunity for us to leverage things like
the "text and mining data" exception,

but, more importantly, to leverage our understanding of the law

and where we can innovate,

where we can remain, at all times, on the right side of the law,

and with a community effort

and developing a community wide coherent approach

where we may seek to influence the law too.

So, the law is ultimately playing catch up a lot
in these technical issues

and therefore there's a real opportunity for us
to assert how the law should behave

and how the law should be,

should reflect what is happening technically.

And that's very much what we're in the midst of at the moment.

But we felt reassured, having gone through this process,

that the risks were within our risk tolerance
and therefore we were comfortable proceeding.

Ultimately, this is a decision that should be taken with a high degree of consideration within your own institution.

I wanted to touch on too some of the other legal and ethical issues associated with these data.

As you'll see later on when Nick showcases our platform and showcases what data we are focusing on, there are a range of things that you can collect from Airbnb.
Those can include pictures of hosts.

So, I mean, these are not necessarily pictures of individuals but rather the thumbnail image that they will have included.

Potentially further information about hosts, what they've included in their "About Me" type statements which may or may not be personally disclosive information.

Physical locations of where the properties themselves are.
So, the data, while we focus on the benign data,

the data that we don’t consider to be disclosive,

there are things in there that one can potentially collect

that may be disclosive,

that may draw questions around

the association with privacy legislation, for example.

There are also ethical questions around

how we approach the practical aspect of
scraping as well, and specifically how we negotiate some of these controls that are intended to limit the effects of nuisance hosts.

We circumvent those by distributing requests across a proxy host network.

And, again, within the copyright law "text and data mining" exception, there are important distinctions drawn between controls intended to protect the copyright of any content that is hosted and any controls or protections that are intended to safeguard the stability and functionality of a given website.
And we effectively interpret the controls that Airbnb have in place as being for the latter.

Our circumvention methods are therefore consistent with the "text and data mining" exception.

But there are legal, well, if not legal questions, there are legitimate questions that may be posed.

Clearly, when you’re doing scraping at scale, there are costs that are being incurred by the data owner themselves.
on their platform, bandwidth costs the most obvious one.

And these are questions that are, at present, not completely resolved.

I'll close in a moment and invite any initial questions that we may have.

But I'll just present this slide here around challenges, that Nick will pick up later on as well.
But these are some of the things that we had to tackle when approaching this particular project.

So, obviously, with a scraping approach,

I mentioned that one of the most overwhelming characteristics of this data collection model is that it's a very one-sided approach. It's not, in any sense, a cooperation.

In that sense, there is a lot of guessing that goes on.

We have no straightforward means of
looking up what data means

and what they represent, the data that we capture.

It's difficult for us to understand the ways that data are published and managed within Airbnb's online environment.

We also have challenges that we need to manage at our own end as well as we get more understanding of the data that's available and more understanding of what the data may or may not represent.

It evolves our specification of our needs.
So, well, more understanding might let us refine our statement of what we actually need.

and then we may have to therefore revise our methods to ensure that we collect more data if and when that need emerges.

Airbnb is obviously not a static resource either. They are just as prone to change and, obviously, technical development and innovations that may take place on their end.
Or, indeed, additional controls to limit the kind of activity that we're involved in that they may consider unattractive.

So, we need to make sure that we are dynamic and aware and keeping up to date with some of these changes as well as, from an expectations management point of view,
acknowledging that there may be ultimately changes that follow

587
00:35:07,089 --> 00:35:08,943
that break everything we've done

588
00:35:09,161 --> 00:35:14,233
or that render our efforts completely undone.

589
00:35:15,268 --> 00:35:16,706
There's also, obviously,

590
00:35:16,996 --> 00:35:21,106
service levels and service limitations

591
00:35:21,206 --> 00:35:24,924
that Airbnb have in place that we have to think about

592
00:35:25,024 --> 00:35:26,582
how we negotiate.

593
00:35:27,613 --> 00:35:29,874
There are limits to the number of results, for example,
that may come out of searches that we rely upon.

So, that obviously impacts how we do things at scale.

And on that same issue of scalability,

it's very important that if, for example, our target is to be able to collect data on a daily basis,

our approach that's designed to do so has to be completed within a single day.
in order to ensure that it's ready to go the following day.

And the last point is around historical data as well.

We're grabbing data that represents the status of the Airbnb platform at a particular time on a particular day.

So, there is no obvious means by which we can capture historical data.

So, we'll look at, for example, how we approach the capture of data from Airbnb calendars,

we want to see on every single individual day
what is the status of the calendar for the forthcoming period of time

in order to then get a better sense of

when things were booked and when those calendars changed and so on.

But we can't get a picture of what that calendar looked like two weeks ago

unless we actually did the scraping activity two weeks ago.

So, in order to build up a historical data set,
the research applications we would have in mind,

00:36:51,581 --> 00:36:53,650
you have to be doing the data collection

00:36:53,750 --> 00:36:54,877
historically as well.

00:36:54,977 --> 00:36:59,097
But Nick will summarise how we tackle some of these in due course.

00:37:02,816 --> 00:37:06,587
The second part of today is very much around the UBDC platform.

00:37:07,376 --> 00:37:10,295
I wanted to mention just a note about how you can get hold of that.

00:37:10,397 --> 00:37:11,533
So, it's on GitHub.

00:37:12,316 --> 00:37:15,776
It is available under the Apache License,
which allows you to distribute, modify, and use,

and we just require the original attribution notices to be preserved.

So, we do require people to obviously acknowledge

the source of this particular platform.

But we're really comfortable with people using it and adapting it to their own data collection needs

goes forward.

So, that was all I wanted to say for the first section.
And I will break now for a moment.

just to invite people to supply questions.

I see that there are a few questions that have already been submitted.

I'll have a wee read through each of these and try and offer a response.

So, the first I have here is, what are the rationale for stakeholders being fearful
or not being collaborative in sharing data sets?

As this would be useful if trying to gain access to other potential data sets.

Well, I mean, that is very much dependent on the individual organisation in question.

I think with this specific example, the researcher colleagues had become aware that Airbnb would be
very resistant to making data available.

And the most obvious assumption in terms of why that would be is that

the type of research that is often being undertaken,

types of associated research, will quite often present

Airbnb and these business models in a relatively negative light

so there is little incentive for them to cooperate with research that

they expect to present them negatively.

I think Airbnb, it's fair to say, are
within a whole range of different locations,

along with other parts of the sharing economy.

Services and companies like Uber.

They do face resistance within particular locations.

The disruptive nature of these business models is such that they do illicit negative responses from some areas.

And I think that there is an assumption that at least some of the outcomes of our research would be to be critical of
these business models.

As you can imagine, there is a whole range of other reasons why companies may be hesitant to make data available.

The other obvious example we see from time to time is companies that are concerned when they engage with academic audiences with data provision, for example, they're concerned that the data may find its way into the hands of competitors.
or it may be used on a commercial basis

that would be to the detriment of themselves.

You can find just as many companies that will say,

well, actually, there's a lot of value in engaging with academia

with data sharing because

it will enhance the data sets,

it will enhance the established reputation
and offer an endorsement of the value of the data that they make available.

But there are many reasons and they’re often unique to the individual business or data owner.

But I think, ultimately, the ideal goal is to try and offer incentives for sharing.

But where you have research that appears to be so contradictory to a particular business model, it’s challenging to see how you would achieve a more cooperative approach.
What happens on IP and copyright law applied to other jurisdictions outside the UK?

Has this been considered and what tips can you provide?

I mean, I should have prefaced all of the coverage of the legal issues.

Obviously, I'm not, myself, a lawyer and nor is this legal advice.

Our understanding though is that, certainly within the European setting,
there is a relatively harmonised copyright regime

and that continues even post-Brexit.

Therefore, there are equivalent "text and data mining" exceptions

across individual legal systems throughout Europe, for example,

which, in some cases, offer even more robust opportunities

for academic researchers.

A lot of the questions around the specific legal advice here though are

quite technical and probably not something we should
spend too much time here on,

nor am I sufficiently well placed to comment authoritatively.

But this issue of the intersection of different legal regimes and the apparent contradictory nature of what they say does present challenges and it presents problems and it has the potential to offer a paralysing effect on researchers who want to do things in good faith.
but they want to do things ethically and on the right side of the law.

One of the key questions here is,

are the data we're collecting from Airbnb themselves covered by copyright law?

If the answer is yes, you may think that means they are more protected and therefore less accessible to us but the converse is actually true because it is only if they are covered by copyright law.
then this exception applies.

Whereas in the case that they are not covered by copyright law,

then this "text and data mining" exception would not apply

because it's not a copyright issue.

So, there are complexities.

There's no precedent that really speaks to the specific circumstances

we're doing or, indeed, general questions around the legality of academic research

using "text and data mining" exceptions
as the lawful basis.

So, we'll not quite wait and see.

What we're keen to do is work with others and work with legal scholars to come up with what we think is a code of conduct and a code of practice that we would like to see becoming manifested within the law and the interpretation of the law.

Next one, can policy really be separated from commercial outcomes?
It's a good question and it's difficult to...

We deal a lot in, obviously, a range of different licenses

and the issue of what is and is not commercial use of

particular data sets is a perennial challenge

and there remains no clear definition that we think is completely water tight.

I think that you're right.

We don't seek to benefit commercially ourselves
and that's perhaps what we would have at the heart of

00:44:32,259 --&gt; 00:44:36,658
our own internal justification for this.

00:44:36,758 --&gt; 00:44:38,447
Or rather, our own kind of

00:44:38,547 --&gt; 00:44:40,436
rationalisation of risk.

00:44:42,132 --&gt; 00:44:47,691
I think, in that respect, the goal of

00:44:50,549 --&gt; 00:44:52,760
the academic sector here is

00:44:53,635 --&gt; 00:44:59,346
not to generate commercial value from these data.

00:44:59,446 --&gt; 00:45:02,213
It may no doubt inevitably lead to
commercial impacts on others

who are operating within those kinds of environments.

But, from our own point of view,

I think we feel relatively reassured that

we’re not establishing ourselves as a competitor to Airbnb

but rather, we are interested in

providing clarity on some of the policy recommendations

and some of the policy impacts of businesses operating like Airbnb are.
If that isn't too fudgy an answer.

How are you not to be seen as a security threat?

I think that that's...

The importance of some of the things Nick will touch on briefly I suppose is how we ensure that we are not interpreted, at least through their automated controls, as being a security threat.

You may argue that our methods are
still kind of akin to what they may expect to see from people who would be security threats.

However, again, we are comfortable with our understanding of what we wish to do with these data and the value that we want to deliver.

So, in that respect, our main priority is just to ensure that we don't set off the technical measures
that would impede our efforts to collect these data.

773
00:46:29,105 --> 00:46:31,542

Even if legally there is no restriction on scraping the content, are there

774
00:46:31,642 --> 00:46:32,941

any legal restrictions on reporting findings?

775
00:46:33,041 --> 00:46:34,404

Well, this is an important point.

776
00:46:35,070 --> 00:46:37,811

And, again, this is something we're

777
00:46:37,911 --> 00:46:41,579

really quite in the dark about.

778
00:46:41,722 --> 00:46:43,729

So, it's a very good point.

779
00:46:43,829 --> 00:46:48,607

So, the "text and data mining" exception is
explicitly around data collection and the use of that data for research.

But it says nothing at all of any...

There's no real clarity about what you can actually do with the results of that research, what you may be able to republish, for example.

So, we can't...

We feel fairly clear that we can't circulate the data that we've collected.

So, instead what we're doing as a data service here is
we're saying let's promote our code, let's promote our methods,

let's encourage other people to do the same kind of thing

or at least give them the technical means

and some sense of the legal understanding

to make a decision to do so.

We can't share the data sets though.

That feels very clear.

What we can even publish within the context of a research article is,
again, not spelled out in clear terms

so we'll ultimately be basing it on

some kind of an informed risk assessment.

And maybe you can make an argument, for instance,

to say that, implicitly, to do academic research work

you're going to publish and implicitly, if you're doing

academic publications within a certain setting,
there are norms, there are expectations around

00:47:53,031 --> 00:47:57,940
what kinds of data should accompany the results within an academic paper.

00:47:59,030 --> 00:48:00,837
That's something we're still wrestling with.

00:48:00,937 --> 00:48:04,072
But, again, it's an unfortunate lack of clarity

00:48:04,172 --> 00:48:08,829
around exactly what one can do

00:48:08,929 --> 00:48:11,195
with the data

00:48:11,500 --> 00:48:15,007
in terms of resharing, in terms of circulating,

00:48:16,204 --> 00:48:19,041
even within the context of
a credible academic publication.

So, this area is mired in some uncertainty and certainly in comparison with a very neat and elegant data licensing agreement where your permitted purpose is spelled out and signed up to, where expectations are explicitly managed and mutually agreed.

This is comparatively untidy, I suppose you could say, this method of data collection.
But it's a means to an end.

that would otherwise be very difficult to achieve.

I'm not exactly sure what the programme for next week, for, beg your pardon, for Thursday's much more research focused coverage is but that is a very pertinent question within that presentation as well.

Not just how do we get this stuff
but, ultimately, we need to be reassured that you can actually do with that data what you wish.

So, this is the main part of the agenda now where we'll see the platform that Nick has developed.

And he will take it away.

So, thanks, Nick.

Hi, all. Good morning.
We are going to demonstrate

this new platform that we've developed here at UBDC

to scrape or capture the data from Airbnb.

The objective of my part here is

basically to demonstrate how we set up a data collection service for Airbnb

and hopefully to kickstart anyone who wishes to do

something similar.

Now, apologies over here because
I want to share the presentation and also the desktop that shows these steps. So, bear with me for a little bit until I figure out how to share screens. Okay. You guys can see my presentation. Hopefully you can. What I was trying to say. It's a little bit of history over here. It's that we've been trying to
capture the data from Airbnb.

And there were several problems.

We initially tried a more direct approach.

And no doubt you're familiar with how the Airbnb website operates.

It's that you get a search page

where you put your query and where you get the data.

Well, something similar which I like to do initially is that

for each search result is go for its listings
and then for its listing to get the data from it

and extract the data from the listing pages

and then check in the database

and hopefully repeat.

There's nothing wrong about this approach initially

but there is, as mentioned before,

Airbnb does have some limitations.
Now really the biggest limitations we could find.

866
00:51:44.789 --> 00:51:48.357
It’s for quality reasons results,

867
00:51:49.142 --> 00:51:53.292
Airbnb only returns 300 results per query

868
00:51:53.982 --> 00:51:56.050
per search. So, they think,

869
00:51:56.347 --> 00:52:02.115
and rightfully so, that no one will go over 300 results

870
00:52:02.422 --> 00:52:06.429
to find their perfect match for their holidays.

871
00:52:06.749 --> 00:52:09.667
Usually, users are finding whatever they want to find

872
00:52:09.767 --> 00:52:12.425
within 50 or 60 results
so they don't bother going any further because it would take too many resources to to find all the listings that would fit into their context.

So, high end results.

So, high end queries, it's out of the question.

For example, Edinburgh, if you put Edinburgh, it has around 10,000 listings or somewhere around there.

So, in the first iteration,
you cannot find through these methods,

you can't find everything.

And if you are familiar with programming,

it's basically one big linear approach over here.

So, if something breaks

and it's within the search results,

then you can't really restart where you picked it up because

every search result is unique every time
that you're searching something,

their engine behind creates a ticket,

and the results are unique for you

based on the day or based on where you are,

based on the many different parameters that they put down.

And even if you were able to go around that,

Airbnb is constantly evolving their model.
They’re constantly putting in new features

or they’re taking out other features.

Like, for example, we do have...

If you’ve recently been to their website,

they reacted to COVID.

Is the listings available for booking during this COVID situation?

Has the UK government rolled out any regulations about COVID?

So, all of them, they need to
be reflected in their website.

So, it's a constant game of mouse and cat here,

where one's changing and the other has to react.

If going for traditional methods,

the biggest problem is that,

not the biggest problem, one of the interesting problems is that after you get the data from the HTML,
you are discarding it, you are throwing it away,

you don't keep it.

So, you can't really compare what you have in the past to what we have in the future,

what you will have in the future.

Or maybe you forgot to

pick up something that you should have

and later on you figure out that, okay, maybe I needed that.
But it's too late.

So, well, having these tried things in mind,

I've similarly looked very carefully at the website.

It's not a static website.

It's basically a responsive web application,

which is a combination itself of

two elements, the HTML element and the JavaScript element.
The HTML code, basically, it's a placeholder, a template,

where it's an initial starting place,

it provides the cards, provides places where you put the information.

And after it's loaded,

it has a JavaScript code in the background

where it kicks in and then starts grabbing extra information

and it sort of hydrates the template
to populate it.

For example, when you load the website

and, afterwards, the JavaScript,

it's requesting extra data from an API endpoint.

For example, what listings there are within this context.

And they get those listings

and then they make sure to create extra HTML elements too.

So, what's been happening.
That's a very interesting concept.

So, basically, one could do...

So, instead of going directly for HTML code,

which is like the end result,

hijack or do the same thing as what the JavaScript does.

Hit these API points

and instead of letting them make the requests for data,

you are making them.
So, if someone is not very familiar with what an API is, basically, it's a well-known application on the web where you're requesting, not requesting, you're hitting them with your web browser and then instead of coming back with an HTML document, they come back with a string which follows a JSON, which is basically a key and a value.
So, you got the ID equals the listing ID,

the number, a listing ID includes a number.

Or is it available? Yes, no.

Is it for what is the price for the day?

That number. And etc.

And imagine that you've got a very, very big stream of data.

It's representing all kinds of information in this matter.

So, the easy way to
talk with Airbnb is basically to use the API.

As mentioned before, you are hitting this endpoint with your browser.

Alternatively, you could do that manually if you put some URLs in your web browser,

you could do that and you would get the appropriate response.

But, alternatively, what you're usually doing within a programming environment is
that you are letting your favourite language construct these requests.

And then your language or your code is creating and generating these requests and they're grabbing that data for you in a programmatical environment.

We have developed, not developed.

We started with...

If you do a little bit of searching on GitHub,
there are multiple libraries that do that for Airbnb. They've not been updated for a long time or they've been abandoned or basically they've been so changed that there is constant evolution or interest of people trying to collect data from Airbnb. Many of them, they have their own limitations,
are a little more current,

but they paint a picture.

And if you are interested in some history,

it's quite interesting.

It shows that, back in the day,

Airbnb had a public API where it was sharing the data.

They decided, at some point, to stop doing that.

But it had published a white paper
that was cataloguing and explaining all their APIs,

the endpoints that they had,

and how you could request for access,

and how they were controlling you.

That think have been deprecated since then.

But people use this information to collect all the data,

all this knowledge into sql databases.

And the one that we started with was
from, this is wrong, it was from another one.

Someone else's effort.

And we extended the model, the library,

to do some of the stuff that we needed.

And, basically, this is...

So, basically, after you implement
	his kind of library,

you can talk directly to Airbnb.
For example, if you’re familiar with Python,

you import the name of the library called Airbnb,

you are creating an API object, and then you get the data.

For example, here’s the new thing that we added.

It’s that you can get the data from this bounding box.

It gives me all the data.

And basically what comes back is a JSON.
This JSON has then been translated to a python dictionary.

And then you take it over with your code

to do whatever you need to do.

And what comes back,

it's something very structured.

This is another endpoint

which gives back information about

the details of the listing IDs.
So, this ID, it has these coordinates,

it’s in this place,

and has all the extra stuff that one might need,

or not.

So, that is at the heart of what we’re trying to do.

It’s that we now have the means of talking to Airbnb.

So, we built a framework, a web, no, not a framework.

A system around that that manages and collects
and scales and puts everything into a more organised way to help collect the data.

So, for our solution over here,

obviously you can...

I'll demonstrate what has been happening.

But if you wanted to do it by yourself,

obviously you will need some kind of prerequirement.

And the biggest one would be Docker.
And since everything, as I like to put it, makes the solution as pain free as possible.

So, if you’re not familiar with Docker, basically what it does is it’s like a mini virtual machines where you are opening them as you see fit, as needed and all the code, including the operating system, the parameters, the configuration or whatnot,
the network.

1052
01:02:04,953 --> 01:02:09,377
Everything is resting inside this little bit of information.

1053
01:02:10,329 --> 01:02:12,244
QGIS, we do need QGIS.

1054
01:02:12,766 --> 01:02:17,954
I'm using it at the moment as a means to communicate with the framework

1055
01:02:18,054 --> 01:02:19,172
from outside.

1056
01:02:19,772 --> 01:02:21,528
Git obviously to...

1057
01:02:22,499 --> 01:02:25,485
QGIS, if you are not familiar with it, is a GIS client,

1058
01:02:25,694 --> 01:02:30,693
basically allows you to create geospatial features
and do some geospatial analysis if you need it.

But we're not going into that.

And Git, obviously, to download the code.

And some other optional stuff that you might find useful.

It is for me.

[DBeaver] It's a database client.

All the data that comes from Airbnb is stored in the database.
So, it's nice that you have something that can visualise and store,

and extract the data later on

and do some kind of transformation.

And a fancy text editor

where, as I said, the response is coming back as JSON.

So, a new text editor allows you

to properly format these strings

so you can more easily understand what is happening here.
Alright. Now, a little bit of architecture.

Or what's been happening.

The whole concept has been...

One of the problems that we had...

Not problems. One of the requirements of the project that we need to finish everything within a certain partner set and amount of time.
So, we are going for the producer-consumer architecture,

which is basically the producer creating tasks

and the consumer is consuming the tasks.

A task, it's basically a message which says

that, in very simple terms, that, okay:

Here is a message with a header which says

you're going to run this command by name

and you are going to use these parameters by name.
And this message is then sent to a broker,

which basically you can parallelize as a post office,

and it's built like a queue.

That queue could contain like a million messages

and they're waiting for someone to pick them up.

So, these queues are going to be consumed by workers

and they're picking up the first message

and they're reading, okay, which function do I have to write, to run?
I have to run this one so which parameter is this parameter?

Okay, leave it to me.

So, the worker will now know

what kind of function to run

and which parameters will go and fetch the data.

And they will consume the data

and then will throw the result information

back to the database,
which then will be available to the framework that we use.

And it shows us a framework, Django, here.

And if you're not familiar with it, it's a web application framework.

It's made, primarily, to create websites.

It's a fully managed solution.

And one of the neat features that it does is

it has its own built-in database management system.
So, it creates tables, it creates databases,

it creates everything that queries the data.

So, everything that gives you all the tools to go ahead,

manage to shuffle through the information that's happening.

So, me, through that framework,

I'm generating the tasks.

So, for example, I'm telling it to give me

all the listings that have not been updated within one day of
the framework queries database.

The database, tells it to gather these 3000 listings that they are stalled when creating the tasks.

And I am giving them to the post office.

The post office then has a whole query queue of 3000 or X number of tasks that they are still there pending.

And the workers will come one by one,
picking it up, and telling them, "Okay, I'm on it",

and they finish it or not.

So, it might sound a little bit complicated to people who have not done something similar to this.

So, as I said, it's...

I will go into something more hands-on.

So, throughout the slides,

we will really demonstrate the following steps on how to install,
configuring UBDC, that platform.

How we can create an area of interest

that we can use for data collection.

And discover listings within that area of interest.

The details for the listings.

Calendar information.

Reviews and maybe booking quotes.
And also, how to configure for scale and production.

Well, without further ado,

let's go and do that.

Obviously, we have...

Now, I'm just going to scroll down that.

And we have set up a virtual machine with

already we have Docker here running.

We've got Git and we've got all the functions entered
and a database connection in postgres.

Obviously, the first step is to...

The first step is always to download the code.

Maybe if you're familiar with GitHub.

If you've done clone, the code is something like that.

So, git clone, the URL that we spoke about before.

And that will clone it to this folder.

I have already done these steps so I will not do it.
In this folder.

And here, we now have all the code that runs, that is on our GitHub.

An exact carbon copy of what we have.

So, the first step, obviously...

as I said before, everything is run through Docker,
the first step is that I'm going to create a database

and the brokers, the post office over here,

which basically are not optional.

It's embedded but it doesn't mean that

you have to use this one.

Maybe you want to use your own database

or a more bespoke data manager you have over there
instead of creating a temporary one like me.

01:09:25,751 --> 01:09:29,447
And I'll say at the end how you can do that.

01:09:29,796 --> 01:09:32,801
But basically you need to change the configuration of

01:09:32,901 --> 01:09:36,319
where the database is but that's not often of concern.

01:09:36,742 --> 01:09:40,127
So, the first step is obviously to create the broker

01:09:40,227 --> 01:09:42,464
and the database.

01:09:42,576 --> 01:09:44,827
And you do that very easily by

01:09:45,127 --> 01:09:49,415
issuing this easy command.
So, Docker compose.

We have the compose files which basically contain all the instructions for what this service should be.

And up or generate these two services.

These two services are the database and the rabbit, which is the broker.

And we press enter.
We are creating it.

The database, it's made.

It creates a spatially enabled database.

And then you can also see the rabbit here,

the broker that has also been activated.

After a little while,

after a little while, it will come down - these are just the logs.

And that is when your code, database, everything is up and running.
I'm opening a new tab

and I'm, cd-ing to the new folders.

cd src. cd Airbnb; test.

And if we go to the database here,

the rabbit, no, DBeaver a database client

that was able to connect to our local

in the port 5432 in the postgres database.
And, basically, it's empty.

It doesn't have any tables.

So, to remedy that,

our application needs a number of tables

that it's going to store the data inside.

To do that,

we need to do the next step.

It's... I'm going to do it here.
So, let's create the tables.

To create the tables by issuing this command.

Basically, what it does, again, it's issuing; telling Docker to run an instance of the worker and through that instance to run the migrate command that will migrate the state of the database.
from the current state
to the final one that has all the tables,
all the end tables.

Excuse me.
So, here we're getting... Okay, okay.
That's basically where we're creating the tables that we need for our operation of this.
If we press this one, we'll see all the tables are here
All of these tables, they are serving a purpose.

But we will explore a little bit.

But, basically, now we have everything.

We have a fully functioning database

with all of the tables that we are supposed to have.

And more importantly, these tables are,

four of them, they are spatially enabled.
Which... Again, apologies.

I need to figure out how to...

Okay. And to close these ones.

More importantly, it's that this database, is spatially enabled, as I was saying.

So, you can connect with it with your favourite GIS client.

And we'll give it a second here

for QGIS to load.
In the meantime...

So, this is the world over here.

One of the things that I have, that is vital over here is

that the...

Yeah, let's go and share them together.

Here the ....

So, the client correctly recognised that

it has four spatial tables here.
This one.

So, one of the things that we want is

the application that is built basically allows you anywhere to

define an area of interest.

If you remove this one.

If you can define the area of interest

by opening QGIS

and then loading the AOI shape table.
And you can draw a polygon

where you say, okay, I'm interested in this location

for this demonstration, I will go to Jersey.

It's a small island with very limited listings,

it's very suitable for our demonstration.

And basically what you do is

you are creating a polygon over here
and you are adding the items that you need.

01:14:46,456 --> 01:14:48,747
Jersey and time stamps.

01:14:48,932 --> 01:14:51,726
You're telling it when you want this GIS.

01:14:51,826 --> 01:14:54,095
This one should be automatically put but it's not.

01:14:54,195 --> 01:14:57,993
And then you are

telling it how these polygons should be used.

01:14:58,491 --> 01:15:00,708
So, are you going to use them to call up the calendars?

01:15:03,100 --> 01:15:06,179
Are you going to use the data to collect the listing details?
Are you going to collect any reviews?

Are you going to use it to discover new listings?

And are you going to use it to collect bookings?

I'm going to use it for that one.

So, basically, you can do it here.

And this is the AOI number one.

We click here. We need this number.

Cannot, no.
But one of the things that I wanted to tell you is that as the application is made, that could work globally, the one thing you need is to define which areas are land-valid- and what are not. So, basically, you need a mask of the land areas and what will filter any kind of requests that will not happen in the sea.
So, you're going to be saving some bandwidth.

You're going to be saving a lot of ...

And to load the mask...

So, again, this one.

To load the mask,

what we are doing is we are downloading the data from GADM.

GADM, if you are not familiar with it,
it's a website from University of California, Berkeley

01:16:20,374 --> 01:16:24,572
where they collect all the world boundaries in certain levels

01:16:24,770 --> 01:16:28,850
and they are creating, and they are giving it to us

01:16:28,950 --> 01:16:32,038
these defined world boundaries

01:16:32,138 --> 01:16:36,776
and that is using that as an input to create the mask.

01:16:36,963 --> 01:16:40,426
And also, seeing as they're kind enough to give data

01:16:40,426 --> 01:16:42,825
to have some information about

01:16:42,925 --> 01:16:45,343
land area you can define for which country.
If you don't put this parameter, the iso parameter, it will do the whole world.

But if you specifically say that I want it for this country, then it will do it for that country.

If you do it for all the countries, if you do it for the whole world, it would take about 10 minutes or so to add all the information.

So, it's not really...
You could do it but we cannot do it this moment.

So, just for Jersey,

I want to add, again, a similar command `docker compose`,

and we're going to log these two definition files.

And I'm going to run an instance for the worker

and they are going to run the command 'load mask'.

And I'm going to tell it to load only Jersey.

So, if I press the enter.
It's downloading the file.

Hopefully, thankfully we have a good internet connection.

And that should only take a couple of minutes.

Yeah, so, then today...

Oh, come on.

Okay.

Today's internet is not very good.
It's starting to wake up.

The file, it's about one gigabyte in size.

It will download the file

and then it will import all of the boundaries for the country we said,

and these boundaries will be used

as a mask.

It's quite important

because otherwise it will just say it cannot understand anything over there
and will filter out the areas that you're trying to harvest.

And we've finished all that.

So, here, we downloaded the file

and we extracted Jersey

and we can see it here as an entry in our world shapefile file.

Excuse me.

So, that knows that Jersey over here is a valid area

so whatever you make over here,
it's going to be processed.

Now, let's change this a little bit.

The display.

Symbology.

Make the fillings transparent so we can see what's happening here.

So, the next step is...

Let's go back to ...
Let's go back to the presentation for a little bit.

So, we defined an area over here.

So, one of the things that is not...

The next step that is happening is that,

if you remember before, it's that we have...

Internally, it's going to generate grids.

And if you remember from before,
one of the things that we added,

01:20:05,159 --> 01:20:09,447
it's the ability to talk to Airbnb through the API

01:20:09,547 --> 01:20:11,899
using coordinates.

01:20:12,026 --> 01:20:16,415
So, we can tell it to give me all the coordinates.

01:20:16,515 --> 01:20:21,294
Sorry, give me all the homes that are within this bounding box.

01:20:21,623 --> 01:20:26,290
So, the next key thing that we've been doing is that

01:20:26,490 --> 01:20:30,438
we are generating grid boxes or bounding boxes

01:20:30,538 --> 01:20:33,746
inside the boundary of this area of interest.
But they follow a specific order, specific logic.

Since we've made for the whole world,

it's one of the logics that we've been using.

it's that we're using the quadkeys.

If you're not familiar with quadkeys,

it's basically a data structure which,

if you take the whole world and divide it by four tiles,

one, two, three, four.
That’s the level one.

If you take the first one and you divide it again into four,

you’ve got 00, 01, 02, 03.

If you take the first one again and you divide it again by three,

you’ve got level three

and you’ve got 000, 001, 002, and 003 and etc.

So, you can go all the way down to level 15

and where you will have these boxes in this level
and you can divide it extra,

you can divide it even more accordingly to your needs.

So, for... That's basically what we're doing.

It's the...

Let's go again with that one.

So, we're going to look at our initial grid.

So, the initial grid is basically like
find me the best grids, quadkeys,

01:21:49,793 -- 01:21:52,431
that they will fit in this bounding box

01:21:52,731 -- 01:21:57,379
and put them into the database.

01:21:58,151 -- 01:22:03,809
So, the code has the logics embedded in that.

01:22:03,909 -- 01:22:06,017
So, basically, you need to kickstart this operation.

01:22:06,117 -- 01:22:12,024
Again, you’re telling it around this worker container

01:22:12,048 -- 01:22:16,675
and prepare me a grid for area of interest one.

01:22:16,775 -- 01:22:19,794
That’s the number one. That’s the area we defined before.
So, if we press the enter, it will do the things.

It gives me some of the initial grids and...

And if we do that there, we will see that initially we have only these two grids.

We'll change this to green.

Green looks okay.

So, these two grids could be used to identify where Airbnb give all the listings for that.
But what seems...

We do have a wee problem.

Like we talked about before,

Airbnb is only giving you back

The listings, not the listings.

The results are limited to the 300 top results.

So, you don't have any guarantees over here that...

These listings, this bounding box will only have
so it's going to be a little bit more abstract.

We are not certain that getting everything that we need to get from there.

So, one of the things we need to do is to divide.

Empirically, I've found that

the search results are coming back faster

if you get like around 50 results, 50 listings.
And the code over here,

1424
01:23:44,458 --> 01:23:50,414
what it does is it asks Airbnb

1425
01:23:50,514 --> 01:23:54,028
how many and only how many listings we have within this bounding box.

1426
01:23:54,128 --> 01:23:55,556
And we've had, like, more than 50.

1427
01:23:55,717 --> 01:23:56,966
Empirical value.

1428
01:23:57,324 --> 01:23:59,014
Divided to its children.

1429
01:23:59,114 --> 01:24:01,562
If you have less than 50, it's fine.

1430
01:24:01,662 --> 01:24:05,140
You leave it like that. And we can do it now.
And hopefully this finishes.

Basically, it's a way of dividing or breaking down the queries into more manageable...

Not the queries, the target where we're going to run the queries, bring it to a level that is more manageable.

So, this could also work in very, very big areas.

So, I do not know if you can see this one but it's really annoying.
So, for example,

you could do something in much bigger areas than Jersey.

This is London.

That I did like a while back.

So, it's grid, imagine...

Not imagine. It's grid, at the time I did it, had

at least, no, at the most 50 listings inside.

So, the interesting thing about this data structure is that
you could use all of these levels as they are and you could cover the whole world without having any overlaps.

This is quite important and interesting.

So, expand, imagine that you are not looking at London, you're not looking at Jersey, you're not looking at London, you're looking at, like, Great Britain, you're looking at the whole of Europe, world, whatever.
The approach will be the same.

If we go back to that.

And if we go to this one.

So, we have sent this one.

Hopefully it's finished. If we move this a little bit.

It did. No, it didn't do it.

Oh, yeah. For this command,

I need to fire up the worker.
Basically, a worker is...

If you remember before, so far, I've been doing the commands in line.

For this one, I sent the work order to the broker and it's waiting for a worker to pick it up and actually do the same thing.

To the worker.

And now... Yeah.

This one.
Yeah, cd src, cd Airbnb.

So, here the worker... It's interesting.

I opened the worker

and the worker understood,

picked out a connection to broker,

and the broker gave it this function,

these tasks to do.

Where is it? Gave them this ID.
And it started doing it.

One of the tasks.

Let's give them like three tasks, as many grids as there are.

So, for each grid up there,

the initial grids, sends how many listings there are.

So, for this listing with this queue.

And this quadkey ID has 108
so it's more than 50.

1486
01:27:08,180 --> 01:27:13,828
Divide it by four and sends for these four new ones,

1487
01:27:13,928 --> 01:27:15,988
if there are more than 50 listings.

1488
01:27:16,088 --> 01:27:18,853
If yes, then divide and move on.

1489
01:27:18,953 --> 01:27:21,168
So, if I move a little bit, you can see that, okay,

1490
01:27:21,268 --> 01:27:24,756
all of these listing, all of these grids, they have less than 50.

1491
01:27:25,692 --> 01:27:31,815
And if we wait for a little bit.

1492
01:27:35,104 --> 01:27:38,206
So, everything...
So, if we open this database,

I think we don't see that...

Okay, these two grids are still pending.

So, for all the ones that we already have,

we have an estimated number of how many listings there are.

I'm saying estimate because there is a way for Airbnb to ask it

and tell you, okay, give me back all the metadata only,

not the actual data.
Metadata means how many listings there are in your query.

and when you did the query with the engine, which is not randomly querying, etc.

But not the actual information about that.

That saves some bandwidth.

So, after we did all the grids.

So, now that we've done all the grids.

Now that we've done all the grids,
we are certain that these ones are,

it's clear that they have less than 50 ones

and now we are ready...

We have an optimised grid network

which, again, contains less than 50.

So, the next step is to find, not find, to tell

the application to find, discover all the listings that we have.
Again, we've sent a command.

01:28:59,125 --> 01:29:03,507
We are telling it using Docker compose.

01:29:03,607 --> 01:29:06,190
Sorry, use the worker container,

01:29:06,290 --> 01:29:08,399
send this task to the workers,

01:29:08,499 --> 01:29:10,434
and tell it to discover all the listings.

01:29:10,534 --> 01:29:11,993
What it's going to do for each grid,

01:29:12,093 --> 01:29:13,893
it's going to run the query

01:29:14,692 --> 01:29:17,031
for this bounding box, give me all the listings,
and then it will extract the information

and it will do some processing and add them to the database.

So, we are issuing the command.

The job is submitted. The ID of the job is this one.

The worker picks it up and it tells us, okay.

This is a problem I am having in this virtual machine

Docker ps.

Docker logs.
Okay. There we are.

So, what has happened over here, for all the grids, it says, okay, find me all the listings in that grid.

And here are the commands coming back.

And for this grid, I found 22.

For this grid, I found 49, and etc.

For this one, I have some duplicates,
I had already seen it zero minutes ago.

And it keeps adding.

So, if we do this.

If we enable the layer with the listings,

we can see all the listings here,

which is very interesting.

Initially, immediately we start to have some kind of sense of
information from Airbnb what’s happening in Jersey.

In Jersey, most of the listings are over here.

They are within this area.

And you can find this route.

Left and right.

Now, there’s questions for post process,

how you’re going to use the data.

Pardon me.
So, this is step one.

Now we have a feeling of how many listings and where the listings are

and, most importantly, we have the listing IDs.

And if you go to the code which says,

demonstrates, describes the API,

how you're going to develop the API,

most of the endpoints of the API designed that
can access the parameter, the listing IDs.

So, you can ask the API,

okay, give me the listing details for

the listing with the listing ID XZ

or, in our case, 21019036,

and that will give you

the listing details, the information about

what this listing contains.
And we can do that as well.

We define an area of interest that we're interested in.

So, we can tell it, okay, give me all the listing details for all the areas data, all the area of interest data we have enabled it to look for that we're interested in getting all the listing details about.

Again, we’re issuing the command.
So, the framework over here

1579
01:32:23,459 --> 01:32:25,774
created a number of tasks.

1580
01:32:27,463 --> 01:32:30,240
Here are all the tasks for the listings. We have a number of listings.

1581
01:32:30,340 --> 01:32:31,769
I don't know how many we have.

1582
01:32:34,519 --> 01:32:35,977
Wait a second, let's count.

1583
01:32:36,757 --> 01:32:37,765
So, if you just count.

1584
01:32:37,865 --> 01:32:43,124
So, we have 200 listings in all of Jersey so far.

1585
01:32:43,635 --> 01:32:45,794
So, we make 200 requests

1586
and these requests for each listing

go to Airbnb and come back with

the information about what's happening with each listing.

Now, if we see here what's coming back.

And now here is the interesting thing.

I do know about the listing details

but what is actually in the...

So, in technical terms,
what I get back is response object.

I am requesting information and the API is responding with some information.

And I’m storing those responses in the database.

And this table, no.

It's the big one.

So, I am storing these responses back into this table,
which has all the interesting things that you guys might be interested in.

And Sublime.

So, if we take the response.

I'm saving it as a JSON, as a document,

and so if you take response over here and you're going to copy the text

that has been JSON and put it,

and I put it here, it's a very, very big string.
So, that’s why I was saying that we need a fancy text editor so that we can format it regularly.

It’s a JSON grid format.

Go back to it.

Here we have interesting things.

Here we have the details about this listing ID.

The listing IDs.

This one has these coordinates,
It's in the city of Jersey in the state of that.

And these are all the photographs that they have.

All this data is used, again,

for Airbnb's website to hydrate their templates

to create the extra features that you see,

to create a responsive website.

Other information that they have.

The host allows you to book it for 21 nights
and they are only allowed to book it for the minimum amount of two nights.

Who the host is, that's their ID, Sarah.

And that's the URL for a picture that she uploaded.

And what kind of bedroom label it has.

A three bedroom place and has these quantities.

Allows pets, no.

Allows events, no.
All the, called a dictionary of data

1633
01:35:32,606 --> 01:35:38,225
with the information that keeps evolving as things are evolving.

1634
01:35:38,664 --> 01:35:40,533
I'm not sure if we have it with COVID.

1635
01:35:40,783 --> 01:35:41,912
No, they haven't done it.

1636
01:35:42,012 --> 01:35:44,400
I've seen some places that...

1637
01:35:47,928 --> 01:35:51,685
I've seen some places with COVID related information

1638
01:35:51,785 --> 01:35:57,054
that you can harvest and service to do some analysis later on.

1639
01:35:58,762 --> 01:36:00,273
But anyhow, yeah.
I've just pressed F5.

So, the same ID, the same thinking is happening.

For these 200...

I have one worker that collects information, the listing details about these 200 listings in Jersey.

And they took some time, over here, to complete.

I mean, you can see this request took two seconds,

this one took another two seconds, that one took three seconds, and etc.
So, if you collect 200,

it would take many, many minutes.

If you had 1000, it would take many, many, hours.

So, one of the things that is interesting is that using this approach, you can see the environment because there are more workers that can work in parallel, make all of the application requests concurrently.

And to do that,
let's call again.

What we would do to collect more...

So, again, I'm telling it instead of having only one worker,

I want to have two workers or let's get three workers or ten.

No, scale please. Yes, thank you.
And give me three workers

01:37:28,870 --> 01:37:31,029
so we can run it a little bit faster.

01:37:32,336 --> 01:37:34,984
So, basically, I'm creating over here...

01:37:37,169 --> 01:37:38,950
So, what I'm creating over here is...

01:37:40,697 --> 01:37:41,716
Come on.

01:37:42,917 --> 01:37:48,115
So, I am putting our two workers over here in the ecosystem

01:37:48,215 --> 01:37:50,092
and all of them, both of them,

01:37:50,781 --> 01:37:52,720
now three of them, they connect to the broker
and they are asking for requests for tasks to run to command.

And all of them, they are doing it individually

without interfering with each other.

There are some other interesting things

that with this technology, with this architecture.

If one of these workers fails for whatever reason,
but I never got it so I assume that he died

so I'm reissuing the task for another alive worker to do.

And, yeah.

Basically, one interesting thing you could do...

Local host.

We have a visual...

Rabbit, the broker, has a web interface that tells us

what's happening in their queues.
So, rabbit, carrots.

All of these. So, here we can see

it's the outstanding tasks that need to happen.

They are pending. So, we still have

another 112 tasks that

they are still in line

until that finishes.
And they are coming back with grids.

01:39:18,763 --> 01:39:20,451
It's been like a long time so...

01:39:21,240 --> 01:39:22,419
Just going to...

01:39:24,656 --> 01:39:26,036
Scale.

01:39:26,136 --> 01:39:28,652
Just put another, make it 10 workers.

01:39:29,612 --> 01:39:30,801
And see it's a bit faster.

01:39:30,901 --> 01:39:32,650
See how...

01:39:36,507 --> 01:39:39,827
So, basically, I'm having 10 concurrent workers
bombarding Airbnb.

Bombarding Airbnb and asking details about everything.

And here, we can see here we've got

the remaining tasks.

They complete much, much faster.

We still have 54 to come back.

Forty-eight.

Thirty-eight.
Just 20.

Yeah, I wasn't expecting it to take that long.

Last time I let this scenario, Jersey didn't have that many listings.

Six.

And we're done.

Well, almost.

And if we open again the responses table over here,
of the listings that we're interested in.

Obviously, getting the listing details,

it's just that we ask for the things that we want to get.

There's particular interest on the calendar.

So, the same idea applies.

We have defined, pardon me.

We have defined our area of interest here,
that we want to use it to collect all the calendars.

01:41:14,607 --> 01:41:18,755
So, again, what we do is

01:41:19,367 --> 01:41:23,186
issue the relevant commands.

01:41:24,404 --> 01:41:24,551
Now, which one?

01:41:29,800 --> 01:41:31,358
Don't stop everything.

01:41:32,698 --> 01:41:33,866
This is very annoying.

01:41:37,765 --> 01:41:39,074
[inaudible].

01:41:44,484 --> 01:41:46,698
Excuse me, folks, for a second here.
Things are happening when I press the button at the wrong time.

I just closed the database.

Here, we've got the database and the rabbit's running, okay.

So, what we want to do is

cd src.
cd Airbnb test.

As I was saying,

we can collect the calendars as well.

Another bit of information.

The calendar, it says for, maybe in context it tells...

We'll give it a look in a little bit.

And, again, we apply for that.

So, we created tasks.
So, the framework it's creating for each
its listing task to get the calendar.
Airbnb is trying to fight with us.
It's 403, I'm not giving you access to it.
So, we're rerouting our query,
response, request through our proxies.
And we defeated it a little bit.
Let's say in quotes.

1757
01:43:25,788 -- 01:43:27,867
And we're getting back the results,

1758
01:43:28,165 -- 01:43:31,123
the responses that contain the calendars.

1759
01:43:31,874 -- 01:43:34,164
And we can see them here.

1760
01:43:35,644 -- 01:43:39,366
The response table is a little bit big.

1761
01:43:39,755 -- 01:43:42,423
So, we got also the type.

1762
01:43:42,632 -- 01:43:43,991
So, the code here for

1763
01:43:45,350 -- 01:43:48,138
type equals CAL.
So, we've got all the information, we've got all the responses for the calendar of that listing ID. If we copy that, let's try to do with this one. Again, copy and paste. JSON pretty. And we've got a nicely documented JSON response that has all the information that one might need.
So, for example, for this listing ID,

if we close it

and open it like this.

It's coming back.

You see, it's a bit hard to navigate here.

I don't know how they do it.

But, basically, what it tells us is that

the listing ID.
Interesting.

So, what it's telling us for this date is the price was 64 units of local currency, which was Great British Pounds, that's what they have in Jersey.

In different countries, they have different currencies.

And I'm sure that you can see that this is quite structured,
you can get the information.

1788
01:45:04,908 --> 01:45:08,141
Although, it's a little bit too much information

1789
01:45:08,241 --> 01:45:13,540
so you might have problems getting value out of it.

1790
01:45:14,349 --> 01:45:18,905
Thankfully, an interesting tip that I can share with you guys,

1791
01:45:19,005 --> 01:45:22,334
it's basically that we could use JSONPath

1792
01:45:22,774 --> 01:45:25,094
to extract the information that you want.

1793
01:45:25,194 --> 01:45:26,851
JSONPath, basically,

1794
01:45:28,159 --> 01:45:31,838
you can give instructions.
There are libraries that can do that with Python,

and libraries that you can give instructions to extract

the information that you want from a document.

Now, let's say that you want

to see all the dates that are available.

There you could run

the appropriate description,

not description, the query
for the library to use that would navigate through the data

and extract all the pieces that you need.

For example, if you wanted the date,

date, okay, you want all the dates,

you tell it double dots, give me all the days

and the top one.

And I want all of them, just them.

And from that one, I just want the date.
Give me all the dates on the right.

And if it's available.

So, you've got a much easier structure

that you can work with.

So, you can divide it by 2 or for this date.

It's available for these dates.

Not available, sorry.
And not available, not available, not available.

1819
01:46:47,766 --> 01:46:49,435
For these dates, it's available, and etc.

1820
01:46:49,535 --> 01:46:51,634
You could do more fun stuff

1821
01:46:52,024 --> 01:46:53,559
to extend the code.

1822
01:46:53,759 --> 01:46:57,337
So, you could go all the way down

1823
01:46:57,437 --> 01:46:59,885
and go to the price node.

1824
01:47:00,963 --> 01:47:01,977
Here it is.

1825
01:47:02,077 --> 01:47:06,226
Here we've got another sub or nested environment.
And from here, on all of that,

1827

I want from here, I want the dates

1828

and I also want the price.

1829

Dates, price.

1830

Dates.

1831

It's local price.

1832

So, for that date, it was £64.

1833

On that date, £95.
That one is £115.

So, you see how as the summer is progressing,

the prices are becoming more steep as the summer is finishing.

The prices are becoming smaller,

lower.

And the lowest price is £64 for that particular listing.

Now, it’s starting to become more interesting over here.

Still, we have not finished all...
1842
01:48:07,221 --> 01:48:08,351
Scale.

1843
01:48:10,764 --> 01:48:12,172
Ten workers.

1844
01:48:14,500 --> 01:48:17,615
Over here so we can finish this one faster, again.

1845
01:48:17,995 --> 01:48:20,463
Let's go back to this management.

1846
01:48:20,563 --> 01:48:22,333
We still have 148.

1847
01:48:24,270 --> 01:48:25,534
What did I want to tell?

1848
01:48:26,125 --> 01:48:28,031
Okay. And now there are listings,
let's talk about another aspect that we could do.

1850
01:48:33,516 --> 01:48:37,625
Another piece of information that I've added here.

1851
01:48:38,135 --> 01:48:39,162
I'm conscious of the time

1852
01:48:39,262 --> 01:48:42,201
so I'm going to show the other interesting thing.

1853
01:48:42,301 --> 01:48:46,740
I think it's the reviews.

1854
01:48:46,956 --> 01:48:49,005
Reviews. Reviews are basically

1855
01:48:50,453 --> 01:48:54,822
reviews about an Airbnb listing

1856
01:48:54,922 --> 01:49:01,034
which contains a review of if it was good or if it was not good.
Also, who made the review.

The reviewee, to whom, and to what listing.

And also, we have this...

We have this approach, no, this functionality built in.

And I could tell if you had to do it for all the listings,

that would take an awful amount of time.

For each review, we also have the guys,

the people that made them,
and we have a different table for that.

We're collecting the data for this as well.

So, instead of telling you the review for one listing,

hopefully it will finish.

And just open a new panel here.

cd src. cd Airbnb, test.

So, this still isn't ideal because I do not know if that exists or not.

So, I'm going to take this one.
Actually, no, it's very remote.

Maybe that'll be interesting.

Take this one.

Copy.

Yes, it does.

It did work.

So, I'm telling it to fetch reviews for
this listing.

1881
01:50:21,444 --> 01:50:23,229
This hasn't understood what I want to tell.

1882
01:50:23,329 --> 01:50:25,221
Again, use the worker, right?

1883
01:50:25,421 --> 01:50:26,785
So...

1884
01:50:28,515 --> 01:50:30,478
That's its worker, okay.

1885
01:50:32,678 --> 01:50:36,023
So, yes, let me...

1886
01:50:36,375 --> 01:50:40,782
So, what it does is it connects to that Airbnb listing.

1887
01:50:43,508 --> 01:50:46,915
It found this number of reviews only.
And if we close that.

I have an extra table for this one so we can go to that listing,
to that table, we can see all the reviews.

So, that's the review ID,
when it was made, when we entered it into the database.

So, the same time, today.

And what's the text? "Brilliant property, perfect for us."

The ability to cook" and etc. etc.
Who made it, for which ID.

Which recipient, to whom.

And what's the response for it.

So, for the reviews, we have two pieces of information.

We have the actual reviews, the text,

which contain information things and in which language it was made.

And who made it.

Hopefully... Oh, yeah.
Okay. What is happening is also that the system, whenever it finds a new user that we have not encountered before, it creates a new task and asks Airbnb to collect data from the user’s API and put them into our database.

And since we have made 10 workers over here, it was fast enough to harvest all these users.
in this small amount of time.

01:52:04,898 --> 01:52:10,355
So, for that, essentially what is happening is that for that listing

we have these 38 people that have interacted with that one.

And that's all the public information that we collect from Airbnb's API.

That's the first name,

what it's about,

how many listings they have.

Some of the users are hosts as well.
Where they say they are from.

That different guy is from London.

And how he has verified himself.

He has verified using an email, phone, and Facebook.

That's what's the picture URL.

We're not storing the picture URLs per se.

We are storing the URL to that picture.

And when he made his profile.
When he added the profile and when was the last time he updated that profile.

And we click onto that one.

Here we can see a photograph of Stephen.

Hopefully, I was allowed to do that.

And, yeah, basically,

then you can collect...

If you have this piece of information, it's quite interesting because

the size of the user for the reviews...
Let's go back. So, you can see when this review was made.

And if you group all these ones,

you can see that in 2000,

the first review was made in 2016

and he has only one review from that year.

For 2017, he had only five reviews.
For 2018, the guy was more popular.

1943
01:53:45,889 --> 01:53:49,684
He had this many for 2019. He had this many for 2020.

1944
01:53:49,784 --> 01:53:51,202
He had only this one, COVID.

1945
01:53:51,302 --> 01:53:54,932
And 2021, at the beginning of the year, he has one.

1946
01:53:55,656 --> 01:54:00,829
And, obviously, you can group by month,

1947
01:54:00,929 --> 01:54:02,935
days, different hours if you want.

1948
01:54:03,035 --> 01:54:06,704
You can go all the way down. You could create interesting graphs

1949
01:54:06,804 --> 01:54:09,172
to see how the reviews... And aggregate that.
Take all the reviews for all the listings for all of Jersey

and see how many reviews they have for 2008, 2020, 2021,

or whatever you want to do.

Basically, that is it from me.

I will skip all this,

all the commands that I put over here.

Oh, yeah, and this bit.

So far, what we've done is we've issued the commands manually.
Manually, give me all the calendars from that area of interest that

I told you that I want the calendars from.

That’s not very convenient if you want a service.

So, what is happening inside,

it also has a scheduler.

Scheduler, it’s a piece of code that

kicks these tasks, the initial tasks,
prespecified every four hours,

specified times, time schedules.

And also, it's embedded in that.

If we activate the mode,

activate the framework,

then a beat more, how it's called, like a heartbeat.

So, I've set it up in the code.
cd src. cd Airbnb explorer dot hear.

1974
01:55:41,012 --> 01:55:44,420
src. dj_airbnb.

1975
01:55:44,650 --> 01:55:46,561
We want the [inaudible].

1976
I'm going to open this folder

1977
01:55:52,037 --> 01:55:53,935
and click this file.

1978
01:55:56,004 --> 01:55:57,958
I might even try to do [inaudible].

1979
01:56:01,718 --> 01:56:03,486
This one. Okay.

1980
01:56:03,975 --> 01:56:05,794
So, what's happening over here is
the database, basically, I'm telling it every four hours,

at minute zero every fourth hour,

run this command.

What this command does is that

it's going to collect all the listings

for the areas that we enabled it.

What this command...

Again, how many? Up to 5000.
If we have like 20,000,

it will only collect the first 5000,

but they are the oldest ones.

What we do here, update the reviews,

similar to that, every four hours.

All the reviews up to 1500,

that they have an age of,

they are older than this many hours,
1997
01:56:49,264 -> 01:56:50,603
that is 14 days.

1998
01:56:52,229 -> 01:56:55,378
So, you define your own schedule here programmatically

1999
01:56:55,478 -> 01:56:57,816
and you’re telling it how you want your service to run.

2000
01:56:57,916 -> 01:56:58,995
How you want it.

2001
01:56:59,983 -> 01:57:02,951
In our case, I mean, I think we’re doing about 80,000,

2002
01:57:03,719 -> 01:57:05,388
90,000 requests per day,

2003
01:57:05,488 -> 01:57:08,607
mostly in a couple of areas that we are interested in

2004
01:57:08,707 -> 01:57:10,732
and we are generating our tasks.

2005
01:57:13,051 --> 01:57:14,739

Let's go back a little bit to my,

2006
01:57:15,769 --> 01:57:19,063

the final bit of our story.

2007
01:57:19,931 --> 01:57:22,222

Go back to the thing.

2008
01:57:22,742 --> 01:57:25,019

We've already talked about scaling.

2009
01:57:25,308 --> 01:57:26,846

Obviously, the other interesting thing

2010
01:57:26,946 --> 01:57:29,906

if you are not familiar with the Docker technology

2011
01:57:30,006 --> 01:57:31,286

and things like Kubernetes, things like that,
and all these containers and things like that,

you're not restricted to running these workers only at your computer.

You could distribute them around the different nodes.

So, you could have hyper scalability.

If your computer over here can only support 100 workers,

that's fine, then buy another, second node,

add it together,

and then you could have 200, 300, 800.
And that’s it for me. That’s my presentation.

The one thing I want to mention is that if you try to do this by yourself, it's going to be very, very hard. Mainly, not because of the code, it's because we, at this moment, in our demonstrations, are using proxy networks, we are using smart proxies through a private company.
Basically, what the smart proxy does is that...

Okay, I'll talk from the beginning. Airbnb, when you are bombarding it, similarly, when it's bombarding it with requests, it starts to throttle you, meaning that it will be alive in the beginning but after three minutes, two minutes or whenever they're putting it,
they’re putting you on their blacklist, let’s say.

They are not...

They are going to

block your requests for a certain amount of time.

And you keep doing it, they’re going to keep blocking you even more.

At some point, you won’t be able to get any information

until you wait, like, a couple of hours

to go back to the normal operations.
So, to go around that,

the only way to go around that is actually to mask your requests

so that they come from different IPs.

And for us, it's the easiest way to do it.

We found a commercial provider that gave us a smart proxy,

which, basically, what it, all our requests are going through them.

And the smart proxy, what it does is,
to do that.

They’re doing that per my requests

through a worker, their worker,

and if the worker gets banned,

then they do it using a different worker with a different IP.

So, what I get back is

only the correct result.

The end result.
That's not 100% bulletproof but we have about over 95% success with that, which I'm very happy with.

So, if you try to do it by yourself, initially you would be, in the large scale especially, initially you would be able to have some successes but don't find yourself surprised
if at some point you're going to have a lot of 404s,

2067
02:00:26,267 --> 02:00:27,325
503s, 501s,

2068
02:00:27,425 --> 02:00:29,743
I don't remember what the error codes are that come back from Airbnb when you fail something.

2069
02:00:29,843 --> 02:00:32,732
And I know that I've spilled over five minutes over time.

2070
02:00:34,348 --> 02:00:38,250
If you have any questions, I'll be happy to answer them.

2071
02:00:38,495 --> 02:00:41,195
I'll be happy to answer them now.

2072
02:00:43,401 --> 02:00:47,263
Thanks, Nick. That's great.
And we have spilled a wee bit over the time

but I thought it was hopefully useful

to let Nick finish with just the last few points there.

We did have one question, just about the API key that we use,

and I already answered that in the chat.

But for everyone else's benefit who didn't see that,

you do not need your own API key for this application.

We actually use Airbnb's own generic API key,
the one that is used by the website itself

in its normal course of operation.

And, as I’ve sort of mentioned in the chat, we’re effectively mimicking

the behaviour of Airbnb’s responsive web application.

And rather than

extracting just the HTML results of

that hydration process that Nick described earlier on.

Rather than grab an HTML text
or, you know, navigating the Document Object Model,

we're just going a little bit further upstream

and grabbing the data from source via the API.

The other question we've received, will you be providing instruction to the walkthrough session so we can have a go later on?

Absolutely. So, what we will circulate in the next day or so will be, obviously, the slides that you've seen.
But, in addition to that,

02:02:01,345 --> 02:02:03,557

a link to, which was actually already within these slides,

02:02:03,657 --> 02:02:07,122

a link to our GitHub page where you can download the code.

02:02:07,222 --> 02:02:13,128

There are README files there that outline the installation steps

02:02:13,292 --> 02:02:17,933

for the, and point to those resources

02:02:18,033 --> 02:02:19,129

that are dependencies.

02:02:19,229 --> 02:02:20,957

So, it will point you to where you can grab Docker

02:02:21,057 --> 02:02:22,605

and all of the things you'll need to know.
Within a Docker container as well as scalability that Nick mentioned.

It also offers us platform independence as well so you can run this on a whole range of different hosts as well.

So, the steps will all be there so you can absolutely have a go and let us know how you get on with that.

But I also invited other questions a few moments ago and there's no other questions that have come in and we are over time just now
so I think I'll probably just draw things to a close.

I'll thank Nick for the presentation

and for the coverage of

what we're really excited about as a data collection platform.

Really keen to hear how other people get on.

And thanks very much for joining us today.

Please, by all means, write to us.

You can contact us at UBDC with any follow up questions
or conversation points.

And, once again, for those of you particularly in research roles,

I would encourage you to register to take part in Thursday's session

as you'll see some of the ways that we're using these data.

And the possibilities that have been outlined today are

finding a really interesting purpose

and a research application.
Have a great day everyone and thanks again, bye bye.