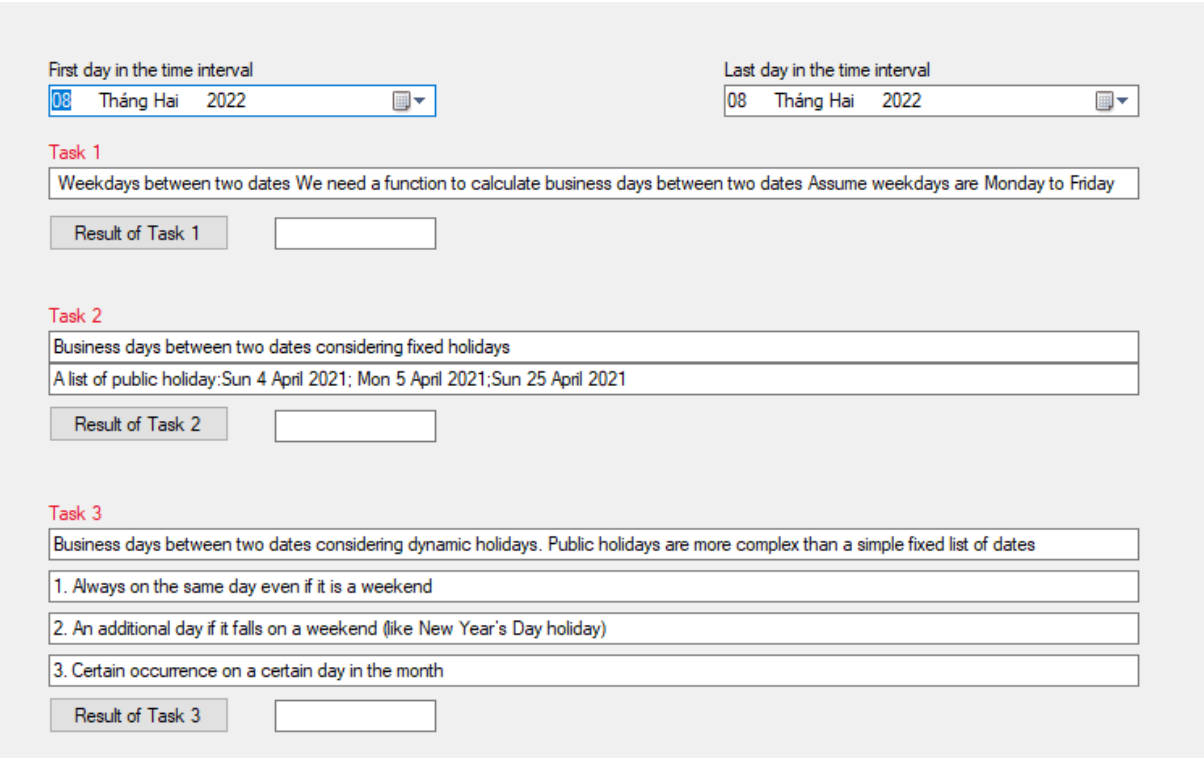
Coding assessment

This assessment is designed to gain further insight into your problem solving, design, and coding skills. Please treat it as an opportunity to demonstrate approaches and design techniques you prefer.

You do not have to complete all the tasks, but it is recommended to do so if possible.

Design Window Form for Task 1, Task 2, Task 3 (Unit Test)



Task 1 Weekdays between two dates

We need a function to calculate business days between two dates

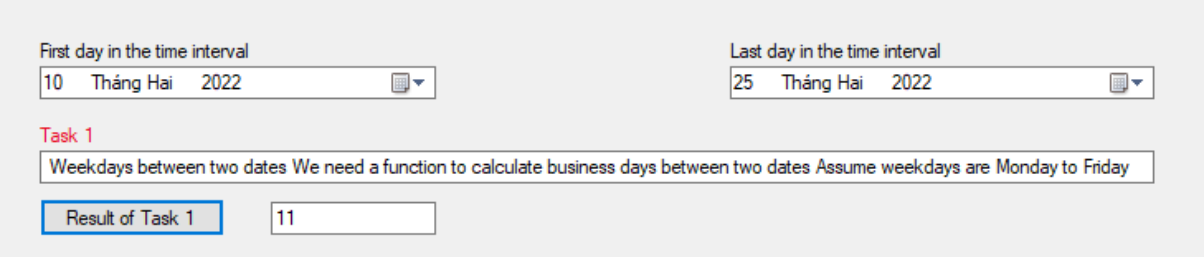
Assume weekdays are Monday to Friday

The returned count should exclude the from date and to date

```
public int GetWorkingDays(DateTime from, DateTime to)
{
    var totalDays = 0;
    for (var date = from; date < to; date = date.AddDays(1))
    {
        if (date.DayOfWeek != DayOfWeek.Saturday
            && date.DayOfWeek != DayOfWeek.Sunday)
            totalDays++;
    }

    return totalDays;
}

private void button1_Click(object sender, EventArgs e)
{
    int count = 0;
    count = GetWorkingDays(dateTimePicker1.Value, dateTimePicker2.Value);
    textBox2.Text = count.ToString();
}
```



Task 2 Business days between two dates considering fixed holidays

Extend the solution to exclude a set of fixed public holidays

Assume weekdays are Monday to Friday

The returned count should exclude the from date and to date

The returned count should exclude dates from a (configurable) list of public holidays

Example:

A list of public holidays:
Sun 4 April 2021
Mon 5 April 2021
Sun 25 April 2021

```
public int GetWorkingDays_subtract_fixedholiday(DateTime from, DateTime to)
    {
      var totalDays = 0;
      DateTime holiday_fix1 = new DateTime(2021, 4, 4);
      DateTime holiday_fix2 = new DateTime(2021, 4, 5);
      DateTime holiday_fix3 = new DateTime(2021, 4, 25);
```

```
        for (var date = from; date < to; date = date.AddDays(1))
        {
            if (date.DayOfWeek != DayOfWeek.Saturday
                && date.DayOfWeek != DayOfWeek.Sunday && date != holiday_fix1 && date
!= holiday_fix2 && date != holiday_fix3)
                totalDays++;
        }

        return totalDays;
    }

private void button2_Click(object sender, EventArgs e)
    {
        int count = 0;
        count = GetWorkingDays_subtract_fixedholiday(dateTimePicker1.Value,
dateTimePicker2.Value);
        textBox4.Text = count.ToString();

    }
```

Task 3 Business days between two dates considering dynamic holidays

Public holidays are more complex than a simple fixed list of dates. Generally, three types occur:

1. Always on the same day even if it is a weekend
2. An additional day if it falls on a weekend (like New Year's Day holiday)
3. Certain occurrence on a certain day in the month

Extend the solution to cater for a public holiday that occurs on a certain day in the month (like Queen's Birthday on the second Monday in June every year)

This should be extensible in future to support the other holiday types

```
public List<DateTime> GetHolidays()
  {
    var client = new WebClient();
    var json = client.DownloadString("https://www.gov.uk/bank-holidays.json");
    var js = new JavaScriptSerializer();
    var holidays = js.Deserialize <Dictionary<string, Holidays>>(json);
    return holidays["england-and-wales"].events.Select(d => d.date).ToList();
  }

  public int GetWorkingDays(DateTime from, DateTime to)
  {
    var totalDays = 0;
    var holidays = GetHolidays();
    for (var date = from.AddDays(1); date <= to; date = date.AddDays(1))
```

```
        {
            if (date.DayOfWeek != DayOfWeek.Saturday
                && date.DayOfWeek != DayOfWeek.Sunday
                && !holidays.Contains(date))
                totalDays++;
        }

        return totalDays;
    }
```

Unit Test

```
[TestClass]
public class WorkingDays
{
    [TestMethod]
    public void SameDayIsZero()
    {
        var service = new WorkingDays();

        var from = new DateTime(2013, 8, 12);

        Assert.AreEqual(0, service.GetWorkingDays(from, from));

    }

    [TestMethod]
    public void CalculateDaysInWorkingWeek()
    {
        var service = new WorkingDays();


var from = new DateTime(2013, 8, 12);
        var to = new DateTime(2013, 8, 16);

        Assert.AreEqual(4, service.GetWorkingDays(from, to), "Mon - Fri = 4");

        Assert.AreEqual(1, service.GetWorkingDays(from, new DateTime(2013, 8, 13)), "Mon
- Tues = 1");
    }

    [TestMethod]
    public void NotIncludeWeekends()
    {
        var service = new WorkingDays();

        var from = new DateTime(2013, 8, 9);
        var to = new DateTime(2013, 8, 16);

        Assert.AreEqual(5, service.GetWorkingDays(from, to), "Fri - Fri = 5");
```

```
        Assert.AreEqual(2, service.GetWorkingDays(from, new DateTime(2013, 8, 13)), "Fri -
Tues = 2");
        Assert.AreEqual(1, service.GetWorkingDays(from, new DateTime(2013, 8, 12)), "Fri -
Mon = 1");
    }

    [TestMethod]
    public void AccountForHolidays()
    {
        var service = new WorkingDays();

        var from = new DateTime(2013, 8, 23);

        Assert.AreEqual(0, service.GetWorkingDays(from, new DateTime(2013, 8, 26)), "Fri -
Mon = 0");

        Assert.AreEqual(1, service.GetWorkingDays(from, new DateTime(2013, 8, 27)), "Fri -
Tues = 1");
    }
}
```

Architecture Example

Task 4 – High-Level Architecture to Ingest Firewall Logs

Document a High-Level Architecture to outline the necessary solution components (using the AWS services) that you would recommend to ingest a High Volume of Network Firewall Logs from 10,000 endpoint firewalls and have the ability to query those logs from a BI Tool.

The framework is based on five pillars:
• Operational Excellence
• Security
• Reliability
• Performance Efficiency
• Cost Optimization

The Well-Architected Framework emphasizes learning, measuring, and improving. It provides a consistent approach for you to evaluate architectures, and implement designs that will scale over time. AWS provides the AWS Well-Architected Tool to help you review your approach prior to development, the state of your workloads prior to production, and the state of your workloads in production. You can compare them to the latest AWS architectural best practices, monitor the overall status of your workloads, and gain insight to potential risks.

You should use tools or services that enable you to centrally govern your environments across accounts, such as AWS Organizations. Services like AWS Control Tower expand this management capability enabling you to define blueprints (supporting your operating models) for the setup of accounts, apply ongoing governance using AWS Organizations, and automate provisioning of new accounts.

On AWS, AWS Systems Manager Change Calendar can be used to record these details. It supports programmatic checks of calendar status to determine if the calendar is open or closed to activity at a particular point of time. Operations activities may be planned around specific “approved” windows of time that are reserved for potentially disruptive activities. AWS Systems Manager Maintenance Windows allows you to schedule activities against instances and other supported resources to automate the activities and make those activities discoverable.

In the AWS shared responsibility model, portions of monitoring are delivered to you through the AWS Personal Health Dashboard. This dashboard provides alerts and remediation guidance when AWS is experiencing events that might affect you. Customers with Business and Enterprise Support subscriptions also get access to the AWS Health API, enabling integration to their event management systems.

On AWS, you can use Amazon CloudWatch Synthetics to create canary scripts to monitor your endpoints and APIs by performing the same actions as your customers. The telemetry generated and the insight gained can enable you to identify issues before your customers are impacted. You can also use CloudWatch Logs Insights to interactively search and analyze your log data using a purpose-built query language. CloudWatch Logs Insights automatically discovers fields in logs from AWS services, and custom log events in JSON. It scales with your log volume and query complexity and gives you answers in seconds, helping you to search for the contributing factors of an incident.

There are multiple ways to automate the execution of runbook and playbook actions on AWS. To respond to an event from a state change in your AWS resources, or from your own custom events, you should create CloudWatch Events rules to trigger responses through CloudWatch targets (for example, Lambda functions, Amazon Simple Notification Service (Amazon SNS) topics, Amazon ECS tasks, and AWS Systems Manager Automation).

Use a data-driven approach to select the patterns and implementation for your architecture and achieve a cost effective solution. AWS Solutions Architects, AWS Reference Architectures, and AWS Partner Network (APN) partners can help you select an architecture based on industry knowledge, but data obtained through benchmarking or load testing will be required to optimize your architecture.

Reach out to AWS for assistance when you need additional guidance or product information. AWS Solutions Architects and AWS Professional Services provide guidance for solution implementation. APN Partners provide AWS expertise to help you unlock agility and innovation for your business

Instances are virtualized servers, allowing you to change their capabilities with a button or an API call. Because resource decisions in the cloud aren’t fixed, you can experiment with different server types. At AWS, these virtual server instances come in different families and sizes, and they offer a wide variety of capabilities, including solid-state drives (SSDs) and

graphics processing units (GPUs). Amazon Elastic Compute Cloud (Amazon EC2) virtual server instances come in different families and sizes. They offer a wide variety of capabilities, including solid-state drives (SSDs) and graphics processing 5 Performance Efficiency Pillar AWS Well-Architected Framework Containers units (GPUs). When you launch an EC2 instance, the instance type that you specify determines the hardware of the host computer used for your instance. Each instance type offers different compute, memory, and storage capabilities. Instance types are grouped in instance families based on these capabilities.

Evaluate available configuration options: Evaluate the various characteristics and configuration options and how they relate to storage. Understand where and how to use provisioned IOPS, SSDs, magnetic storage, object storage, archival storage, or ephemeral storage to optimize storage space and performance for your workload. Amazon EBS provides a range of options that allow you to optimize storage performance and cost for your workload. These options are divided into two major categories: SSD-backed storage for transactional workloads, such as databases and boot volumes (performance depends primarily on IOPS).

Forecasting future AWS costs: Forecasting enables finance stakeholders to set expectations with other internal and external organization stakeholders, and helps improve your organization’s financial predictability. AWS Cost Explorer can be used to perform forecasting for your cost and usage.

DevOps Example

Task 5 – ExampleCorp's marketing department wants a public-facing Wordpress blog.

Your task as a DevOps Engineer is to build the automation that will be used to deploy Wordpress as well as any necessary supporting infrastructure.

You can host Wordpress in any way that you'd prefer, for example directly on an EC2 instance, or within a container on ECS or EKS.

The tools we would like for him to use are Terraform and Ansible:

We expect to run this on an actual AWS account, so be sure to tell us:

- what tools we need
- what, if any, configuration is needed (eg: AWS Profile)
- how to deploy

Creating AWS EC2 Instances with Terraform

EC2 instances are defined using the terraform.tfvars, some values (ami, vpc_security_group_ids and subnet_id) are derived from modules output so the definition is in the aws_ec2_pro_wp.tf file as terraform.tfvars doesn't allow interpolation.

```
#-----------------------
# WP PRO
#-----------------------
aws_ec2_pro_pub_wp_01 = {
name = "ditwl-ec2-pro-pub-wp01"
ami = "" #Uses data.aws_ami.ubuntu1604.id
instance_type = "t2.micro" #AWS Free Tier: 750 hours per month of Linux, RHEL, or SLES t2.micro instance usage
availability_zone = "us-east-1a"
key_name = "ditwl_kp_infradmin"
# vpc_security_group_ids = SEE TF file
# subnet_id = SEE TF file
associate_public_ip_address = true
root_block_device_size = 8
# See https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSVolumeTypes.html
root_block_device_volume_type = "gp2"
tag_private_name = "ditwl-ec2-pro-pub-wp-01"
tag_public_name = "www"
tag_app = "wp"
tag_app_id = "wp-01"
tag_os = "ubuntu"
tag_os_id = "ubuntu-16"
tags_environment = "pro"
tag_cost_center = "ditwl-permanent"
tags_volume = "ditwl-ec2-pro-pub-wp-01-root"
}
#-----------------------
# WP PRO Security Group
#-----------------------
aws_sg_ec2_pro_pub_wp_01 = {
sec_name = "ditwl-sg-ec2-pro-pub-01"
sec_description = "ditwl - WP server access rules - Pub, Env: PRO"
```

```
allow_all_outbound = false
}
aws_sr_ec2_pro_pub_wp_01_internet_to_80 = {
type = "ingress"
from_port = 80
to_port = 80
protocol = "tcp"
cidr_blocks = "0.0.0.0/0"
description = "Access from Internet to port 80"
}
aws_sr_ec2_pro_pub_wp_01_internet_to_443 = {
type = "ingress"
from_port = 443
to_port = 443
protocol = "tcp"
cidr_blocks = "0.0.0.0/0"
description = "Access from Internet to port 443"
```

All EC2 instance names and its Security Rules and Groups follow a naming pattern:

- **Cloud:** a prefix specifying the unique name of this cloud across all available clouds and providers. In this case the prefix will be: ditwl that stands for **D**emo **ITW**onder **L**ab in lowercase.
- **Resource**: a short name identifying the resource, in this case:
  - **ec2**: for EC2 Instances
  - **sg-ec2**: for an EC2 Security Group
  - **sr-ec2**: for an EC2 Security Rule
- **Environment**: for resources that are not to be shared between environments, a 3 letter acronym for the environment:
  - **pro**: production
  - **pre**: preproduction
  - **dev**: development
- **Visibility**: for resources that can be either public or private, a 3 letter acronym for the visibility:
  - **pub**: for public resources
  - **pri**: for private resources
- **Name/ID**: optional a name or ID that describes the usage of the resource or the number of the resource instance, for example:
  - **EC2 resource number**: this will be an EC2 instance for WordPress **wp** number **01** for Public Zones and Pro Environment. We might have new EC2 instances for WP in the future, adding a number now will make it easier to grow the infrastructure later on.
  - **EC2 security group number**: this will be the EC2 security group **01** for Public Zones and Pro Environment for **wp**. We might have new security groups on the future, adding a number now will make it easier to grow the infrastructure later on.
  - **Description of the purpose of the rule**: using a description like **instances_to_instance_port** explains the intended usage of the rule. In this case the rules allow access from the Internet to ports 80 and 443. (as we are not using a Load Balancer, Internet connections are directly to instances)

Private Key for the EC2 Instance

In order to access the created Linux instances in AWS you will need an SSH client. Authentication will use a private key, and in the case of Ubuntu a username named “ubuntu”.

The private key needs to be registered in AWS EC2 console, it can be uploaded to the console or created using a wizard.

1. Visit the AWS console
2. Select the region where instances will be created (as Key Pais are unique to each region),
3. Go to EC2 AWS web console
4. Go to Network & Security and Key Pairs.
5. Create a new Key Pair and name it ditwl_kp_infradmin. AWS generates a PEM file that you should store in a safe place.
6. Save the downloaded pem file in ${HOME}/keys/ditwl_kp_infradmin.pem. It will be used by Ansible in the next tutorial.

Launching AWS EC2 Instances with Terraform

Having configured most of the values for the instance in the terraform.tfvars, now the file aws_ec2_pro_wp.tf makes use of Terraform modules to create the resources.

The Terraform module **/modules/aws/ec2/instance/add** is used to create the EC2 instance. Most of the variables come from the aws_ec2_pro_pub_wp_01 variable definition from terraform.tfvars and the rest are interpolations to other resources:

- **ami** = “${data.aws_ami.ubuntu1604.id}”. Uses the output from the data source aws_ami. See Route 53 and AMI Lookup for an explanation.
- **disable_api_termination** = “${var.is_production ? true : false}”. Uses the value from is_production variable to prevent EC2 instance destroy. See Avoiding AWS instance destroy with Terraform for an explanation.
- **vpc_security_group_ids** = [“${module.aws_sec_group_ec2_default.id}”,”${module.aws_sg_ec2_pro_pub_wp_01.id}”] is a list of security groups that will allow access to the instance.
- **register_dns_private** = true and **register_dns_public** = true are used for Instance registration in private and public DNS. See Registering EC2 instances in Route 53 for an explanation.
    - # Create WP instance
    - module "aws_ec2_pro_pub_wp_01" {
    - source = "./modules/aws/ec2/instance/add"
    - name = var.aws_ec2_pro_pub_wp_01["name"]
    - ami = data.aws_ami.ubuntu1604.id
    - instance_type = var.aws_ec2_pro_pub_wp_01["instance_type"]
    - availability_zone = var.aws_ec2_pro_pub_wp_01["availability_zone"]
    - key_name = var.aws_ec2_pro_pub_wp_01["key_name"]
    - disable_api_termination = var.is_production ? **true** : **false**
    - vpc_security_group_ids = [module.aws_sg_ec2_default.id,module.aws_sg_ec2_pro_pub_wp_01.id]

- subnet_id = module.aws_sn_za_pro_pub_32.id
- associate_public_ip_address = var.aws_ec2_pro_pub_wp_01["associate_public_ip_address"]
- instance_tags = {}
- tag_private_name = var.aws_ec2_pro_pub_wp_01["tag_private_name"]
- tag_public_name = var.aws_ec2_pro_pub_wp_01["tag_public_name"]
- tag_app = var.aws_ec2_pro_pub_wp_01["tag_app"]
- tag_app_id = var.aws_ec2_pro_pub_wp_01["tag_app_id"]
- tag_os = var.aws_ec2_pro_pub_wp_01["tag_os"]
- tag_os_id = var.aws_ec2_pro_pub_wp_01["tag_os_id"]
- tags_environment = var.aws_ec2_pro_pub_wp_01["tags_environment"]
- tag_cost_center = var.aws_ec2_pro_pub_wp_01["tag_cost_center"]
- register_dns_private = **true**
- route53_private_zone_id = module.aws_route53_public.id
- register_dns_public = **true**
- route53_public_zone_id = module.aws_route53_public.id
- root_block_device = {
- volume_size = var.aws_ec2_pro_pub_wp_01["root_block_device_size"]
- volume_type = var.aws_ec2_pro_pub_wp_01["root_block_device_volume_type"]
- delete_on_termination = var.is_production ? **false** : **true** #If production, Do not delete!
- }
- volume_tags = {
- Name = var.aws_ec2_pro_pub_wp_01["name"]
- }
- ignore_changes = ["ami"]
- }

Resource Security in AWS with Terraform

Securing AWS VPC resources with Terraform makes use of 3 modules:

- **modules/aws/security/group**: creates a security group and returns the id to be used in rules and instance association.
- **modules/aws/security/rule/cidr_blocks**: creates a security rule associated to a previously created security group, the cidr_blocks module uses an IP address range as source for the traffic.
- **modules/aws/security/rule/source_group**: creates a security rule associated to a previously created security group, thesource_group module uses another security group as source. See RDS creation for an example of its usage.
    - A **generic default group for each resource type**: these groups are used to hold default groups that apply to the type of resource, for example the SSH access to EC2 instances from a fixed administration IP address or the access to the database port for administration from a fixed administration IP.
      Examples:

    - **ditwl-sg-rds-mariadb-def**: default security group for all the MariaDB RDS resources. It is specific for MariaDB RDS resources as other type of database will use a different port.
    - **ditwl-sg-ec2-def**: default security group for all EC2 instances. It will have the SSH access rule.
- **A specific group for each resource:** It is recommended to have a group for each resource, it will be named using part of the resource name and the prefix will have the cloud and the resource type. Examples:
    - **ditwl-aws-sg-rds-mariadb-pro-pub-01**: security group for all the MariaDB RDS resources.
    - **ditwl-sg-ec2-pro-pub-01**: security group for all the WP EC2 instances.

Avoid creating too many groups and don't use CIDR as a source (except for Internet as a source). It is better to use groups as a source, that way an element gets access to other resources by being a member of a group, not by having a specific IP that can change.