Logger for Cloud connector Go

Hello Everyone! In this page we would try to understand zap (Blazing fast, structured, leveled logging package for Go, see how it compares with other famous packages here) and how it is implemented at Avi.

As a prerequisites please make sure you have the following packages:-

- zap `go get -u go.uber.org/zap`
- lumberjack `go get gopkg.in/natefinch/lumberjack.v2`

We'll learn it all with an example!

Example

```
package main

import (
        "log"

        "go.uber.org/zap"
        "go.uber.org/zap/zapcore"
        "gopkg.in/natefinch/lumberjack.v2"
)

func logger1() {
        // In contexts where performance is nice, but not critical, use
the SugaredLogger. It's 4-10x faster than other structured logging
packages and supports both structured and printf-style logging.
SugaredLogger's structured logging APIs are loosely typed and accept a
variadic number of key-value pairs.
        logger, err := zap.NewDevelopment()
        if err != nil {
                log.Fatalf("can't initialize zap logger: %v", err) //
Replace this line with something as we dont want our program to stop
        }
        sugar := logger.Sugar() // Thus we see that it is easy to convert
to a SugaredLogger, do sugar.Desugar() to get back normal logger
        defer sugar.Sync()      // By default, loggers are unbuffered.
However, since zap's low-level APIs allow buffering, calling Sync before
letting your process exit is a good habit.

        // Info uses fmt.Sprint to construct and log a message
        // Infof uses fmt.Sprintf to log a templated message.
        // Infow logs a message with some additional context. The variadic
key-value pairs are treated as they are in With.
        // Read more about all this here:
https://godoc.org/go.uber.org/zap#SugaredLogger.Infow
        sugar.Infow("This is an INFO message with fields",
                "region", "us-west", // key-value pair 1
                "id", 2, // key-value pair 2
```

```
        )
        // In the rare contexts where every microsecond and every
allocation matter, use the Logger. It's even faster than the SugaredLogger
and allocates far less, but it only supports strongly-typed, structured
logging.
        // As mentioned simply before, logger doesn't have Infof nor Infow
        logger.Info("This is an INFO message with fields",
zap.String("region", "us-west"), zap.Int("id", 2))
}

// expected output:
// 2019-05-13T16:55:05.141+0530    INFO    tutorial/prodex1.go:22  This is
an INFO message with fields     {"region": "us-west", "id": 2}
// 2019-05-13T16:55:05.141+0530    INFO    tutorial/prodex1.go:22  This is
an INFO message with fields     {"region": "us-west", "id": 2}
// Also I wrote NewDevelopment (as this is the style we will mostly stick
with) but there are as well NewProduction and NewExample

func logger2() { // Custom Logger (This way I understand to be the best
among others)
        atom := zap.NewAtomicLevel()
        cfg := zap.Config{
                Encoding: "console", // Other setting is json
                Level:    atom,      // very important concept!
                // We have four levels, viz. Debug, Info, Warn, Error,
Fatal. Putting DebugLevel here will imply that we can log all of these 5
levels, if I were to put ErrorLevel, then I would only be able to log
Error, Fatal level. This setting can be dynamically changed using our atom
as shown below!
                // Note: Logging at Fatal will cause the program to
terminate.
                OutputPaths:      []string{"stdout",
"/Users/sourabhaggarwal/Desktop/some.log"}, // can put multiple paths!
                ErrorOutputPaths: []string{"stderr"},
                InitialFields: map[string]interface{}{ // Initial fields
                        "some": "bar"},
                EncoderConfig: zapcore.EncoderConfig{
                        MessageKey: "message",

                        LevelKey:    "level",
                        EncodeLevel: zapcore.CapitalLevelEncoder,

                        TimeKey:    "time",
                        EncodeTime: zapcore.ISO8601TimeEncoder,

                        CallerKey:    "caller",
                        EncodeCaller: zapcore.ShortCallerEncoder,

                        LineEnding: "\n----*-----\n",
                        // Although I didn't enable but we can as well
have stack trace!
                },
        }
        atom.SetLevel(zap.DebugLevel)
```

```
        logger, _ := cfg.Build()
        defer logger.Sync()
        logger.Debug("Debug!")
        logger.Warn("Info will get ignored")
        atom.SetLevel(zap.WarnLevel) // This will make sure that info and
below doesn't get logged.
        logger.Info("This is an INFO message with fields",
zap.String("region", "us-west"), zap.Int("id", 2))
}

// Output:
// 2019-05-13T17:19:41.839+0530    DEBUG   tutorial/prodex1.go:65  Debug!
{"some": "bar"}
// ----*-----
// 2019-05-13T17:19:41.839+0530    WARN    tutorial/prodex1.go:66  Info
will get ignored   {"some": "bar"}
// ----*-----

func logger3() { // Now lets add lumberjack (helps in log rotation)
        atom := zap.NewAtomicLevel()
        atom.SetLevel(zap.DebugLevel)
        w := zapcore.AddSync(&lumberjack.Logger{
                // put you desired file path here!
                Filename: "/Users/sourabhaggarwal/Desktop/logName",
                // MaxAge is the maximum number of days to retain old log
files based on the
                // timestamp encoded in their filename.  Note that a day
is defined as 24
                // hours and may not exactly correspond to calendar days
due to daylight
                // savings, leap seconds, etc. The default is not to
remove old log files
                // based on age.
                MaxAge:  28, // days
                MaxSize: 20, // megabytes
                // MaxBackups is the maximum number of old log files to
retain.  The default
                // is to retain all old log files (though MaxAge may still
cause them to get
                // deleted.)
                MaxBackups: 3,
                Compress:   true, // Yes we want to compress the files
        })
        core := zapcore.NewCore(
                // could have used here zap.NewDevelopmentEncoderConfig()
but still not getting line numbers...
                zapcore.NewConsoleEncoder(zapcore.EncoderConfig{ // same
old settings available
                        MessageKey: "message",

                        LevelKey:    "level",
                        EncodeLevel: zapcore.CapitalLevelEncoder,

                        TimeKey:    "time",
```

```
                        EncodeTime: zapcore.ISO8601TimeEncoder,

                        CallerKey:    "caller",
                        EncodeCaller: zapcore.ShortCallerEncoder,

                        LineEnding: "\n----*-----\n",
                }),
                w,
                atom,
        )
        logger := zap.New(core, zap.AddCaller())
        defer logger.Sync()
        sugar := logger.Sugar()
        sugar.Infow("This is an INFO message with fields", "region", "us-
west", "id", 2)
        // logger.Info("This is an INFO message with fields",
zap.String("region", "us-west"), zap.Int("id", 2))
}

// Output is in your mentioned file!

func main() {
        logger1()
        logger2()
        logger3()
}
```

Implementation at Avi

- It's almost the same as ***logger3*** just that as we have wrapped everything inside a package, thus, it is therefore required to modify

  `logger := zap.New(core, zap.AddCaller())`

  to

  `logger := zap.New(core, zap.AddCaller(), zap.AddCallerSkip(1))`

  So as to show ***that*** caller which is invoking the suitable method of our package.

- Due to simplicity of `SugaredLogger` ***(as evident in*** `logger1` ***where its details are as well explained)***, we have decided to stick with it.

To see it head over to avi-dev/go/src/avi/infra/avilog.go